УДК 681.518
ББК 32.973.202
И 74

Автор-составитель В. В. Бондарева, канд. техн. наук, доцент

Рецензенты: В. Е. Быховцев, д-р техн. наук, профессор
кафедры вычислительной математики
и программирования Гомельского государственного
университета им. Ф. Скорины;
Т. А. Заяц, ст. преподаватель кафедры
информационно-вычислительных систем
Белорусского торгово-экономического университета
потребительской кооперации

Рекомендован научно-методическим советом учреждения образования «Белорусский торгово-экономический университет потребительской кооперации». Протокол № 5 от 28 декабря 2010 г.

И 74 Информационные **системы и технологии. Web-программирование на JavaScript : практикум к лабораторным занятиям для студентов специальности 1-26 03 01 «Управление информационными ресурсами» / авт.-сост. В. В. Бондарева. – Гомель : учреждение образования «Белорусский торгово-экономический университет потребительской кооперации», 2011. – 104 с.**
ISBN 978-985-461-890-6

УДК 681.518
ББК 32.973.202

ISBN 978-985-461-890-6

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

В практикуме рассматриваются вопросы курса «Информационные системы и технологии».

Данное издание посвящено изучению инструментария для создания Web-приложений на языке JavaScript. JavaScript – объектно-ориентированный скриптовый язык программирования. Он обычно используется как встраиваемый язык для программного доступа к объектам приложений. Наиболее широкое применение находит в браузерах как язык сценариев для придания интерактивности Web-страницам.

В практикуме рассмотрено назначение, применение и синтаксис языка JavaScript, использование основных методов и свойств объектов JavaScript.

По изучаемым темам приведен теоретический материал, примеры скриптов, результаты их выполнения и задания для самостоятельной работы. Практикум содержит список рекомендуемой литературы, который позволит обеспечить расширение спектра знаний по изучаемым вопросам.

Лабораторная работа 1
ВНЕДРЕНИЕ ОБЪЕКТОВ JAVASCRIPT В HTML-ДОКУМЕНТ

1.1. Назначение и применение JavaScript

В 1995 г. специалисты компании Netscape создали механизм управления страницами, разработав язык программирования *JavaScript*. Это язык управления сценариями просмотра гипертекстовых страниц Web на стороне клиента. Интерпретатор языка встроен в Web-браузер.

JavaScript – объектно-ориентированный язык программирования. Для создания механизма управления страницами на клиентской стороне было предложено использовать объектную модель документа – DOM (*Document Object Model*) (рисунок 1).

Объектная модель документа не является частью языка JavaScript. Строго говоря, это интерфейс прикладного программирования для представления документа (например, документа HTML), обеспечения доступа к его элементам и интерактивного изменения их свойств. Более того, DOM предоставляет механизмы для изменения самой структуры документа (добавление и удаление элементов, изменение их содержимого). Все версии языка JavaScript в той или иной степени поддерживают объектную модель документа.

Более подробно объектная модель документа будет рассмотрена далее.

1.2. Внедрение объектов JavaScript в HTML-документ

Создавать программы JavaScript можно с помощью простого текстового редактора (например, Блокнота) или с помощью программ, специально предназначенных для разработки Web-сайтов (например, FrontPage или Dreamweaver).

Программы (сценарии) JavaScript встраиваются в документ HTML. Рассмотрим три способа использования JavaScript внутри HTML-документа.

Первый способ

Программа JavaScript может быть встроена в тело документа HTML при помощи тегов `<script>` и `</script>`. С помощью тега `<script>` можно встроить в документ сценарий, составленный на языке JavaScript, VBScript или PHP. Язык указывается с помощью параметра `language`.

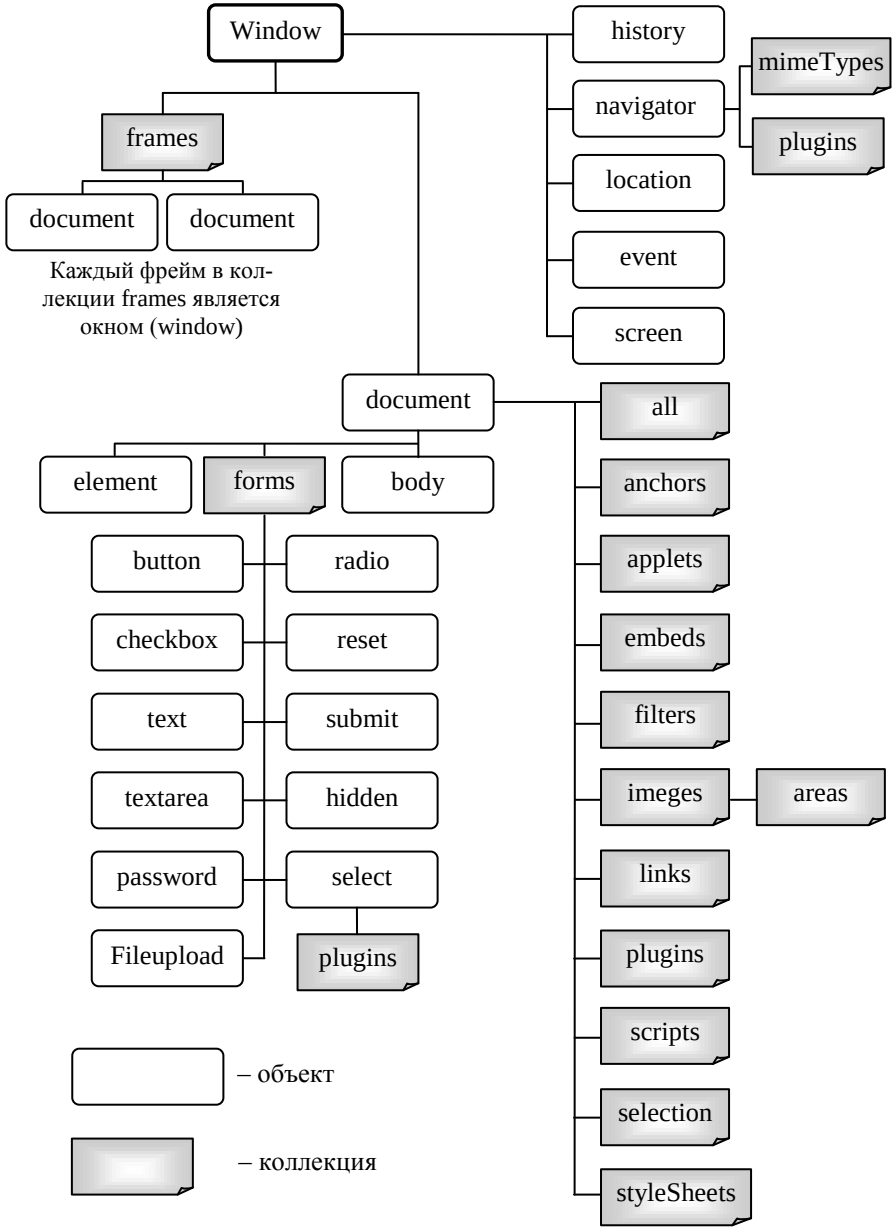


Рисунок 1 – **Объектная модель документа**

Старые браузеры, появившиеся раньше JavaScript, игнорируют теги <script> и </script>, а все, что находится между ними, интерпретируют как содержимое HTML-документа. Результат может быть самым неожиданным. Поэтому текст сценария оформлен как комментарий с применением операторов <!-- и -->. Перед символами --> дописывают символы //. Это позволяет обеспечить работоспособность сценария в различных браузерах. Некоторые из них (Netscape Navigator) в сценариях JavaScript рассматривают строку --> как ошибочную. Символы // используются в JavaScript для выделения комментариев и предписывают браузерам игнорировать символы, записанные после них (в том числе и -->).

Для обозначения комментариев можно использовать также конструкцию /*...*/. Этот способ удобен, если комментарий содержит несколько строк.

Пример 1. Вставка программой JavaScript предложения «Hello, world!» в документ HTML (листинг 1, рисунок 2).

Листинг 1

```
<html>
<head>
<title>Hello, world!</title>
</head>
<body>
<h1>JavaScript Test</h1>
<script language="JavaScript">
<!--
document.write("Hello, world!");
// -->
</script>
</body>
</html>
```

Программа простая и содержит только одну строку на JavaScript:

```
document.write("Hello, world!");
```

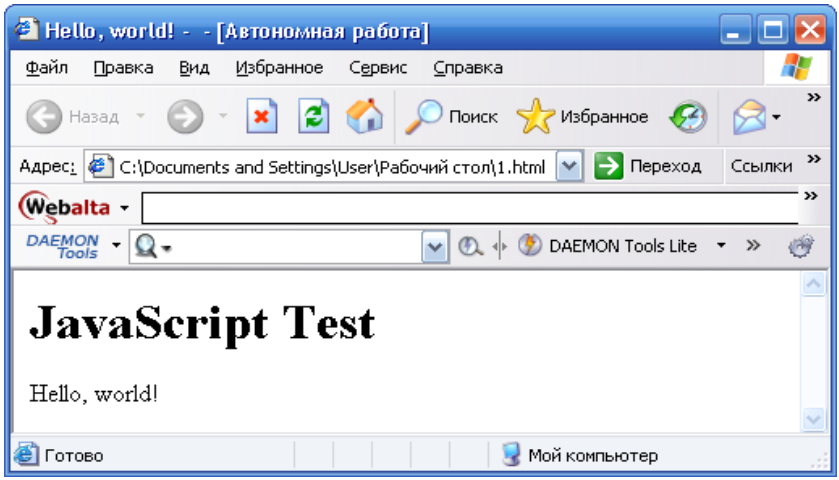


Рисунок 2 – **Результат выполнения листинга 1**

Здесь для объекта с именем `document` вызывается метод `write`. В качестве параметра ему передается текстовая строка «Hello, world!». Строка закрывается символом точка с запятой, хотя этот символ может и отсутствовать. Объект `document` – это документ HTML, загруженный в окно браузера. Он содержит в себе объекты, свойства и методы, предназначенные для работы с элементами этого документа HTML, а также для взаимодействия с другими объектами. Метод `write` записывает в тело документа HTML приветственную строку «Hello, world!». При этом документ будет выглядеть так, как будто эта строка находится в нем на месте сценария, как в примере 2.

Пример 2. Создание документа HTML (листинг 2).

Листинг 2

```
<html>
<head>
<title>Hello, world!</title>
</head>
<body>
<h1>JavaScript Test</h1>
Hello, world!
</body>
</html>
```

Второй способ

Исходный текст любого сценария должен включаться в документы HTML. Однако есть техническая возможность оформлять программы на JavaScript в отдельных файлах (расширение `*.js`) и в страницах HTML указывать на эти файлы ссылки. Браузер, загружая оформленные подобным образом HTML-документы, загружает оформленные в отдельных файлах сценарии и подставляет их вместо соответствующих ссылок. Такой способ включения JavaScript-сценариев удобен, если один и тот же сценарий должен быть включен во множество документов HTML, или же если есть необходимость скрыть исходный код от просмотра пользователями.

Ссылки на файлы с подгружаемыми скриптами оформляются с помощью параметра `src` тега `<script>`, допускающего указывать адрес URL файла сценария. Следующий пример демонстрирует использование параметра `src`.

Пример 3. Включение в исходный текст документа HTML ссылки на файл сценария `hello.js` (листинг 3).

Листинг 3

```
<html>
<head>
<title>Hello, world!</title>
</head>
<body>
<h1>JavaScript Test</h1>
<script language="JavaScript" src="hello.js">
</script>
</body>
</html>
```

Ссылка оформлена с применением тегов `<script>` и `</script>`, однако между этими операторами нет ни одной строчки исходного текста. Этот текст перенесен в файл `hello.js` (листинг 4).

Листинг 4

```
document.write("<HR>")
document.write("Hello, world!")
document.write("<HR>")
```

В параметре `src` примера 3 задано только имя файла, так как он

находится в том же каталоге, что и ссылающийся на него файл документа HTML. Однако можно указать и относительный путь, и полный адрес URL, например:

```
<script language="JavaScript"
src="http://www.myserver.by/scripts/hello.js">.
```

Третий способ

В сценариях JavaScript активно применяют функции. Они, как правило, определяются в области заголовка документа, выделенной операторами `<head>` и `</head>`, с помощью ключевого слова `function` и оформляются с применением операторов `<script>` и `</script>`.

Синтаксис:

```
function имя_функции ([параметры])
{
код
}
```

Кроме того, в теле документа HTML должен быть еще один раздел сценариев, выделенный аналогичным образом для вызова функции.

Синтаксис:

```
имя_функции ([параметры])
```

Пример 4. Использование функции в тексте программы. В области заголовка документа HTML определяется функция с именем `pr`. Эта функция вызывается из сценария, расположенного в теле документа, и выводит в документ HTML текст (листинг 5).

Листинг 5

```
<html>
<head>
<title>Hello, world!</title>
<script language="JavaScript">
<!--
function pr()
{
document.write("Hello, world!")
}
// -->
```

```
</script>
</head>
<body>
<h1>JavaScript Test</h1>
<script language="JavaScript">
<!--
pr()
// -->
</script>
</body>
</html>
```

Интерпретация документа HTML и встроенных в него сценариев происходит по мере загрузки документа. Поэтому если в сценарии одни функции вызывают другие, то их определения необходимо разместить выше вызывающих. Размещение определений функций в разделе заголовка документа гарантирует, что они будут загружены до момента загрузки тела документа.

Задание для самостоятельного выполнения

Выполните скрипты, приведенные в листингах 1–3, 5.

Лабораторная работа 2
ВВОД И ВЫВОД ДАННЫХ

В JavaScript предусмотрены довольно скудные средства для ввода и вывода данных. Это оправдано, так как JavaScript создавался, в первую очередь, как язык сценариев для Web-страниц, а основой Web-страниц является HTML-код, который специально рассчитан на форматирование информации и создание пользовательского интерфейса.

Для ввода и вывода данных можно воспользоваться тремя стандартными методами: `alert()`, `prompt()` и `confirm()`.

2.1. Метод `alert()`

Метод `alert()` позволяет выводить диалоговое окно с сообщением и кнопкой *OK*.

Синтаксис:
`alert(сообщение)`

Сообщение представляет собой данные любого вида: последовательность символов, заключенную в кавычки, число (в кавычках или без), переменную или выражение.

Диалоговое окно, выведенное методом `alert()`, можно убрать, щелкнув по кнопке *ОК*. До тех пор пока это не сделано, переход к ранее открытым окнам не возможен. Окна, обладающие свойством останавливать все последующие действия пользователя и программ, называются модальными. Окно, создаваемое с помощью метода `alert()`, является модальным.

Пример 5. Использование метода `alert()` (листинг 6, рисунок 3).

Листинг 6

```
<html>
<head>
<title>Hello, world!</title>
<script language="JavaScript">
<!--
function pr()
{
alert("Hello, world!")
}
// -->
</script>
</head>
<body>
<h1>JavaScript Test</h1>
<script language="JavaScript">
<!--
pr()
// -->
</script>
</body>
</html>
```

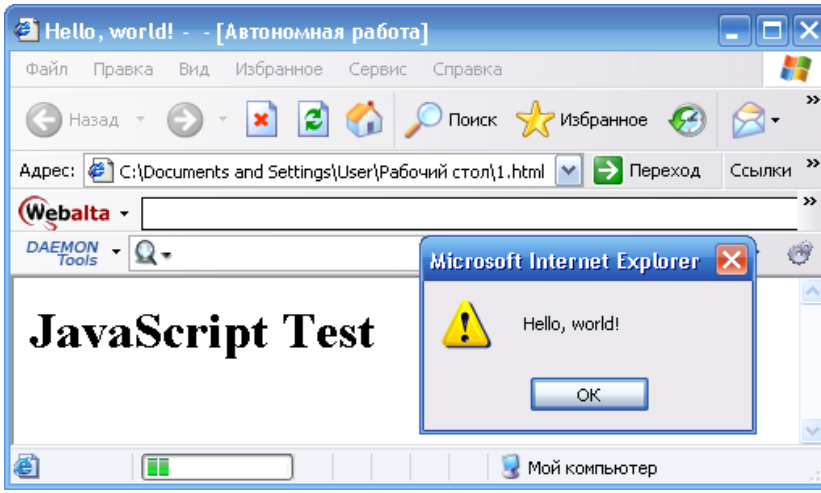


Рисунок 3 – **Вывод данных методом `alert()`**

2.2. Метод `confirm()`

Метод `confirm()` позволяет вывести диалоговое окно с сообщением и двумя кнопками – *ОК* и *Отмена*. В отличие от метода `alert()`, этот метод возвращает логическую величину, значение которой зависит от того, на какой из двух кнопок щелкнул пользователь. Если щелчок был на кнопке *ОК*, то возвращается значение `true`, если на кнопке *Отмена*, то `false`. Возвращаемое значение можно обработать в программе и создать эффект интерактивности (диалогового взаимодействия программы с пользователем).

Синтаксис:
`confirm(сообщение)`

Сообщение, также как и в методе `alert()`, представляет собой данные любого вида.

Диалоговое окно, выведенное методом `confirm()`, можно убрать, щелкнув по любой из двух кнопок *ОК* или *Отмена*. Окно, создаваемое с помощью метода `confirm()`, также является модальным.

Пример 6. Использование метода `confirm()` (листинг 7, рисунок 4).

Листинг 7

```
<html>
<head>
<title>Hello, world!</title>
</head>
<body>
<script language="JavaScript">
<!--
confirm("Уверены, что хотите войти?")
// -->
</script>
</body>
</html>
```

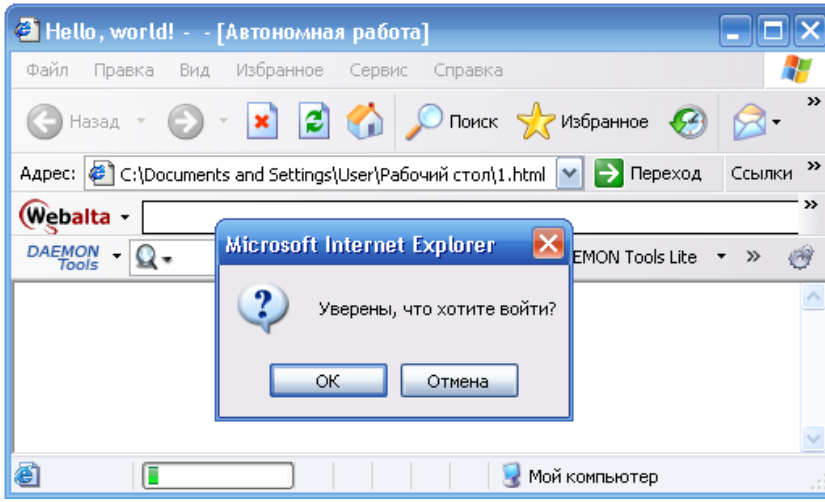


Рисунок 4 – **Вывод данных методом `confirm()`**

2.3. Метод `prompt()`

Метод `prompt()` позволяет вывести на экран диалоговое окно с сообщением и текстовым полем, в которое пользователь может ввести данные. Кроме того, в этом окне предусмотрены две кнопки – *OK* и *Отмена*. Метод `prompt()` принимает два параметра: сообщение и значение, которое должно появиться в текстовом поле ввода данных

по умолчанию. Если пользователь щелкнет на кнопке *ОК*, то возвращается значение `true`, если на кнопке *Отмена*, то `false`. Возвращаемое значение, также как и в методе `confirm()`, можно обработать в программе.

Синтаксис:

```
prompt(сообщение, значение поля ввода данных)
```

Параметры метода `prompt()` не являются обязательными. Если необходимо, чтобы поле ввода было пустым, то второй параметр должен быть определен как пустая строка `""`. Например:

```
prompt("Введите имя","").
```

Диалоговое окно, выведенное методом `prompt()`, можно убрать, щелкнув по любой из двух кнопок *ОК* или *Отмена*. Окно, создаваемое с помощью метода `prompt()`, также является модальным.

Пример 7. Вывод в окне браузера введенной в диалоговом окне текстовой строки (листинг 8, рисунки 5 и 6).

Листинг 8

```
<html>
<head>
<title>Hello, world!</title>
<script language="JavaScript">
<!--
function pr()
{
HelloStr=prompt("Введите приветственное
сообщение:", "")
document.write(HelloStr);
}
// -->
</script>
</head>
<body>
<h1>JavaScript Test</h1>
<script language="JavaScript">
<!--
pr()
```

```
// -->
</script>
</body>
</html>
```

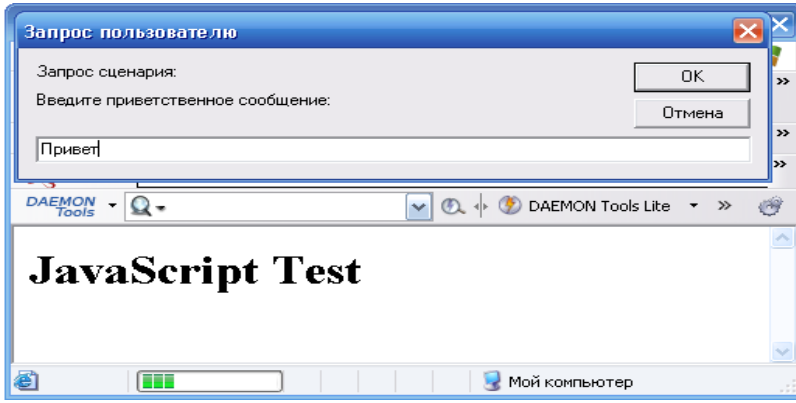


Рисунок 5 – **Вывод данных методом `prompt()`**

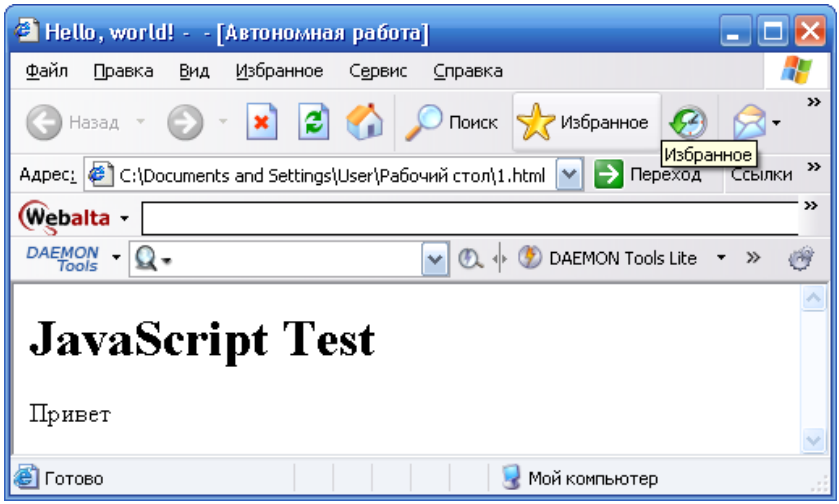


Рисунок 6 – **Результат выполнения листинга 8**

Задания для самостоятельного выполнения

1. Выполните скрипты, приведенные в листингах 6–8.

2. Напишите скрипт, который выводит на экран окно сообщения с кнопкой *OK*, а при нажатии на кнопку *OK* в окне браузера пишет фразу «Hello, world!».

3. Напишите скрипт, который запрашивает у пользователя информацию, а затем выводит ее в окне сообщения.

Лабораторная работа 3
БАЗОВЫЕ КОНСТРУКЦИИ ЯЗЫКА JAVASCRIPT

3.1. Типы переменных

В JavaScript существует несколько различных типов переменных, которые определяются типом хранимых в них данных. Переменные различных типов могут хранить в качестве значений разнообразную числовую и текстовую информацию. В частности, в JavaScript допустимы переменные, представленные в таблице 1.

Таблица 1 – **Типы переменных в JavaScript**

Тип переменных	Пример	Описание значения
Строковый или символьный (string)	`"Привет"` `"тел. 412–3456"`	**Последовательность символов, заключенная в кавычки, двойные или одинарные**
Числовой (number)	`3.14` `–567` `+2.5`	**Число, последовательность цифр, перед которой может быть указан знак числа («+» или «–»). Перед положительными числами не обязательно ставить знак «+». Целая и дробная части чисел разделяются точкой. Число записывается без кавычек**
Логический или булевый (boolean)	`true` `false`	**`True` (истина) или `false` (ложь); возможны только два значения**
Null	–	**Отсутствие какого бы то ни было значения**
Объект (object)	–	**Программный объект, определяемый своими свойствами. В частности, массив также является объектом**
Функция (function)	–	**Определение функции**

JavaScript – слаботипизированный язык. Это означает, что переменные можно использовать без явного объявления их типа, и комбинация различных типов переменных не приводит к ошибке.

Слабая типизированность языка означает также, что при создании новой переменной нет необходимости задавать тип переменной. Достаточно ввести имя переменной, а ее тип будет установлен автоматически в зависимости от типа значения, присвоенного этой переменной.

3.2. Преобразование строковых значений в численные

При создании программ на JavaScript за типами данных следит программист. Если он перепутает типы, то интерпретатор не зафиксирует ошибки, а попытается привести данные к некоторому типу, чтобы выполнить указанную операцию.

Для преобразования строк в числа в JavaScript предусмотрены встроенные функции `parseInt()` и `parseFloat()`.

Синтаксис функции `parseInt()`:

```
parseInt(строка, основание)
```

Функция `parseInt()` преобразует указанную в параметре строку в целое число в системе счисления по указанному основанию (8, 10 или 16). Если основание не указано, то предполагается 10, т. е. десятеричная система счисления. При преобразовании в целое число округления не происходит: дробная часть просто отбрасывается.

Пример 8. Использование функции `parseInt()`:

```
parseInt("3.14") // 3
parseInt("-7.876) // -7
parseInt("435") // 435
parseInt("Вася") // NaN, строка не является числом
```

Синтаксис функции `parseFloat()`:

```
parseFloat(строка)
```

Функция `parseFloat()` преобразует указанную строку в число с плавающей разделительной точкой.

Пример 9. Использование функции `parseFloat()`:

```
parseFloat("3.14") // 3.14
parseFloat("-7.875") // -7.875
parseFloat("435") // 435
parseFloat("Вася") /* NaN, строка не является
```

```
числом */
parseFloat("17.5") // 17.5
```

3.3. Имена переменных

Переменная предназначена для хранения данных. Данные, сохраняемые в переменной, называют значениями этой переменной. Имя переменной – это последовательность букв, цифр и символа подчеркивания без пробелов и знаков препинания, начинающаяся обязательно с буквы или символа подчеркивания.

Примеры правильных имен переменных:

`myFamily`, `my_adress`, `_x`, `tel412_3456`.

Примеры неправильных имен переменных:

`512group`, `my adress`, `tel:412 3456`.

При выборе имен переменных нельзя использовать слова, используемые в определениях конструкций языка. Например, нельзя выбирать слова `var`, `if`, `else`, `true` и ряд других. Имя переменной чувствительно к регистру, например, `str` и `STR` – разные переменные.

3.4. Создание переменных

Создать переменную в программе можно несколькими способами:

1. С помощью оператора присвоения значений в следующем формате:

```
имя_переменной = значение.
```

Пример 10. Создание переменной с помощью оператора присвоения:

```
myName = "Иван"
```

2. С помощью ключевого слова `var` в следующем формате:

```
var имя_переменной.
```

В этом случае созданная переменная не имеет никакого значения,

но может его получить в дальнейшем с помощью оператора присвоения.

Пример 11. Создание переменной с помощью ключевого слова `var`:

```
var myName
myName = "Иван"
```

3. С помощью ключевого слова `var` и оператора присвоения в следующем формате:

```
var имя_переменной = значение.
```

Пример 12. Создание переменной с помощью ключевого слова `var` и оператора присвоения:

```
var myName = "Иван"
```

3.5. Операторы

Операторы предназначены для составления выражений. Оператор применяется к одному или двум данным, которые называются операндами. Элементарное выражение, состоящее из операндов и оператора, вычисляется интерпретатором и имеет некоторое значение, т. е. оператор возвращает значение, которое можно присвоить переменной.

3.5.1. Арифметические операторы

Арифметические операторы, такие как сложение, умножение и т. д., в JavaScript могут применяться к данным любых типов (таблица 2).

Таблица 2 – **Арифметические операторы в JavaScript**

Оператор	Название	Пример
+	**Сложение**	`X + Y`
–	**Вычитание**	`X – Y`
*	**Умножение**	`X * Y`
/	**Деление**	`X / Y`
%	**Деление по модулю (возвращает остаток от деления первого числа на второе)**	`X % Y`

Оператор	Название	Пример
++	**Увеличение на 1, инкремент**	**X++ (эквивалентно выражению X + 1)**
--	**Уменьшение на 1, декремент**	**Y-- (эквивалентно выражению Y – 1)**

Пример 13. Использование арифметических операторов:

```
y = 5 // значение переменной y равно 5
x = y + 3 // значение переменной x равно 8 (5 + 3)
x++ // значение переменной x стало равным 9 (8 + 1)
x-- // значение переменной x стало равным 8 (9 - 1)
y++ // значение переменной y равно 6 (5 + 1)
5 - 3 // значение равно 2
```

В случае строковых данных оператор сложения дает в результате строку, полученную путем приписывания справа второй строки к первой. Для строк оператор сложения называется оператором конкатенации.

Пример 14. Использование арифметических операторов для строковых данных:

```
x = "Вася" // значение переменной x равно "Вася"
y = "Маша" // значение переменной y равно "Маша"
z = x + " " + y /* значение переменной z равно
"Вася Маша" */
z = x + 5 // значение переменной z равно "Вася5"
s = "20" + 5 // значение переменной s равно "205"
```

3.5.2. Дополнительные операторы присвоения

Существует несколько разновидностей оператора присвоения, сочетающих в себе действия обычного оператора «=» и операторов сложения, вычитания, умножения, деления и деления по модулю (таблица 3).

Таблица 3 – **Дополнительные операторы присвоения в JavaScript**

Оператор	Пример	Эквивалентное выражение
+=	X += Y	X = X + Y

-=	**X -= Y**	**X = X - Y**
***=**	**X * = Y**	**X = X * Y**
/=	**X /= Y**	**X = X / Y**
%=	**X %= Y**	**X = X % Y**

Пример 15. Использование дополнительных операторов присвоения:

```
x = "Вася"
x += ", привет" // значение переменной x равно
"Вася, привет"
```

3.5.3. Операторы сравнения

В программах часто приходится проверять, выполняются ли какие-либо условия. Например, является ли браузер пользователя Microsoft Internet Explorer или Netscape Navigator. В зависимости от результата процесс дальнейших вычислений может проходить по-разному.

Проверяемые условия формируются на основе операторов сравнения «=», «<», «>» и т. п. (таблица 4). Результатом вычисления элементарного выражения, содержащего оператор сравнения и операнды (сравниваемые данные), является логическое значение. Если условие выполняется, то возвращается `true`, иначе возвращается `false`.

Таблица 4 – **Операторы сравнения в JavaScript**

Оператор	Название	Пример
==	**Равно**	**X == Y**
!=	**Не равно**	**X != Y**
>	**Больше**	**X > Y**
>=	**Больше или равно**	**X >= Y**
<	**Меньше**	**X < Y**
<=	**Меньше или равно**	**X <= Y**

Сравнивать можно числа, строки и логические значения. Сравнение чисел происходит по правилам арифметики, а строк – путем сравнения ASCII-кодов символов начиная с левого конца строк.

Пример 16. Использование операторов сравнения:

```
"abcd"=="abc" // возвращает false
```

```
"abcd"=="abcd" // возвращает true
"abcd"==" abed" /* возвращает false (1-й символ
2-го операнда - пробел) */
```

3.5.4. Логические операторы

Логические данные, обычно получаемые с помощью элементарных выражений, содержащие операторы сравнения, можно объединять в более сложные выражения. Для этого используются логические операторы – И и ИЛИ, а также оператор отрицания НЕ (таблица 5). Выражения с логическими операторами возвращают значение `true` или `false`.

Таблица 5 – **Логические операторы в JavaScript**

Оператор	Название	Пример
!	**Отрицание (НЕ) применяется к одному операнду, изменяя его значение на противоположное**	**!X**
&&	**И**	**X&&Y**
||	**ИЛИ**	**X||Y**

3.6. Оператор условного перехода

В языке JavaScript предусмотрены операторы условного перехода `if` и `switch`, которые позволяют выполнять разные программные строки в зависимости от условия. Рассмотрим оператор `if`.

Синтаксис:

```
if(условие)
  строка 1
[else
  строка 2]
```

Часть оператора, выделенная квадратными скобками, является необязательной.

Пример 17. Вывод диалогового окна с тем или иным сообщением в зависимости от значения переменной `age` (листинг 9).

Листинг 9

```
<html>
<head>
<title>Совершеннолетие</title>
</head>
<body>
<script language="JavaScript">
<!--
age=prompt("Укажите свой возраст","")
if (age < 18)
{
alert("Вы не достигли совершеннолетия")
}
else
{
alert ("Подтвердите свое совершеннолетие")
}
// -->
</script>
</body>
</html>
```

3.7. Операторы цикла

Оператор цикла обеспечивает многократное выполнение блока программного кода до тех пор, пока не выполнится некоторое условие. В JavaScript предусмотрены три оператора цикла: `for`, `while` и `do...while`.

3.7.1. Оператор `for`

Оператор `for` – это оператор со счетчиком циклов.

Синтаксис:

```
for([инициализация]; [условие]; [итерация])
{
  строки тела цикла
}
```

В области инициализации выполняется присваивание начальных значений переменным цикла. Здесь допустимо объявление новых пе-

ременных при помощи ключевого слова `var` (например, `i = 0` или `var i = 0`). Область условия задает условие выхода из цикла, например, `i <= 5`. Это условие оценивается каждый раз при прохождении цикла. Если в результате оценки получается логическое значение `true`, выполняются строки тела цикла. Область итерации применяется для изменения значений счетчика цикла, например, `i = i + 1` или `i++`.

Пример 18. Вычисление суммы всех целых положительных чисел от 1 до 15 (листинг 10).

Листинг 10

```
<html>
<head>
<title>Сумма чисел</title>
</head>
<body>
<script language="JavaScript">
<!--
var s = 0
for (i = 1; i <= 15; i++)
{
s = s + i
}
alert("Сумма чисел "+ s)
// -->
</script>
</body>
</html>
```

3.7.2. Оператор `while`

Для организации итерационных циклов с предусловием используется оператор `while`.

Синтаксис:

```
while(условие)
{
  строки тела цикла
}
```

Если в результате оценки условия получается значение `true`, тогда итерация выполняется, если `false` – цикл прерывается.

Пример 19. Вычисление `n!` (листинг 11).

Листинг 11

```
<html>
<head>
<title>Факториал</title>
</head>
<body>
<script language="JavaScript">
<!--
var z = 1
n=prompt("Введите n","")
if (n > 1)
{
i = 2
while (i <= n)
{
z = z * i // z хранит результат n!
i++
}
}
alert("Факториал "+ n+" равен "+z)
// -->
</script>
</body>
</html>
```

3.7.3. Оператор ***break***

С помощью оператора `break` можно прервать выполнение цикла, созданного операторами `for` или `while`, в любом месте.

Пример 20. Конструкция с использованием оператора `break`:

```
var i = 0
while(true)
{
  ...
  i++
```

```
  if(i > 10)
    break
  ...
}
```

Задания для самостоятельного выполнения[1]

1. Выполните скрипты, приведенные в листингах 9–11.
2. Напишите скрипт, который запрашивает два числа и выводит в окне браузера наибольшее из них.
3. Напишите скрипт, который запрашивает число и выводит соответствующее количество раз фразу «Hello, world».
4. Напишите скрипт, запрашивающий количество учащихся студенческой группы и в соответствии с этим количеством – их фамилии и имена. Фамилии и имена необходимо отобразить в окне браузера.
5. Доработайте предыдущее задание для вывода списка учащихся в виде таблицы с заголовком, шапкой и первым столбцом, в котором указаны порядковые номера (*).
6. Напишите скрипт, который запрашивает номер месяца и выводит название времени года. Номер месяца и название времени года необходимо отобразить в окне браузера.
7. Доработайте предыдущее задание так, чтобы номер месяца запрашивался до тех пор, пока пользователь в окне ввода не нажмет кнопку *Отмена* (*).
8. Напишите скрипт, выводящий в окно браузера таблицу умножения без форматирования.
9. Доработайте предыдущее задание для вывода таблицы умножения в виде таблицы с заголовком, шапкой и первым столбцом, в котором указаны множители. Цифры в шапке и первом столбце выведите синим цветом (*).
10. Доработайте пример по вычислению факториала для случая, если пользователь введет любое значение `n` (положительное, отрицательное или текстовое).

Лабораторная работа 4
ВНУТРЕННИЕ ОБЪЕКТЫ JAVASCRIPT

Объекты представляют собой программные единицы, обладающие некоторыми свойствами. Программный код встроенных в JavaScript

[1] Звездочкой помечены задания повышенной сложности.

объектов недоступен.

Математические вычисления, сложная обработка строк и дат, а также создание массивов в JavaScript производятся с помощью соответствующих встроенных объектов.

Внутренние (встроенные) объекты имеют фиксированные названия, свойства и методы. Свойства аналогичны обычным переменным. Они имеют имена и значения. Некоторые свойства объектов доступны только для чтения, другие свойства доступны и для записи – их значения можно изменять с помощью оператора присвоения.

Методы аналогичны функциям, они могут иметь или не иметь параметры. Можно применить метод к объекту.

Синтаксис:

```
имя_объекта.метод([параметры])
```

Для разработчиков Web-сайтов особенно важны следующие объекты: `Array` (массивы), `Date` (работа с датами), `Math` (математические формулы и константы) и `String` (обработка строк).

4.1. Объект `Array`

Объект `Array` представляет собой упорядоченный набор данных, который называется массивом. Количество элементов в массиве называется длиной массива. В JavaScript автоматически создаются индексы для элементов массива, поэтому к элементам массива можно обращаться по индексам. Нумерация начинается с нуля.

Примерами объектов-массивов в браузере служат гиперсвязи, метки, формы, фреймы. Массив создается с использованием ключевого слова `new`.

Синтаксис:

```
имя_массива = new Array ([длина_массива])
```

Здесь `длина_массива` является необязательным числовым параметром. Если длина массива не указана, то создается пустой массив, не содержащий ни одного элемента. В противном случае создается массив с указанным количеством элементов, однако все они имеют значение `null`, т. е. не имеют значений.

Создав массив, можно присвоить значения его элементам, используя для этого оператор присвоения.

Пример 21. Создание массива с помощью конструктора `Array()` и вывод значений на экран (листинг 12).

Листинг 12

```
<html>
<head>
<title>Массивы</title>
</head>
<body>
<script language="JavaScript">
<!--
myArray = new Array()
myArray[0]= "first element"
myArray[1]= "second element"
myArray[2]= "third element"
for (var i = 0; i < 3; i++)
{
document.write(myArray[i] + "<br>")
}
// -->
</script>
</body>
</html>
```

Можно также создать массив, определив его элементы в круглых скобках за ключевым словом `Array`.

Пример 22. Другой способ создания массива с помощью конструктора `Array()` и вывод значений на экран (листинг 13).

Листинг 13

```
<html>
<head>
<title>Массивы</title>
</head>
<body>
<script language="JavaScript">
<!--
myArray = new Array("first element",
```

```
"second element","third element")
for (var i = 0; i < 3; i++)
{
document.write(myArray[i] + "<br>")
}
// -->
</script>
</body>
</html>
```

У объекта `Array` есть свойство `length`, значением которого является длина массива.

Синтаксис:

```
имя_массива.length
```

Пример 23. Определение длины массива (листинг 14).

Листинг 14

```
<html>
<head>
<title>Hello, world!</title>
</head>
<body>
<script language="JavaScript">
<!--
myArray = new Array()
myArray[0]= "first element"
myArray[1]= "second element"
myArray[2]= "third element"
a=myArray.length
document.write("длина массива myArray " + a)
// -->
</script>
</body>
</html>
```

Методы объекта `Array` предназначены для управления данными, сохраненными в структуре массива.

Основные методы объекта **`Array`**

1. Метод `concat()` осуществляет конкатенацию массивов – объединяет два массива в третий. Данный метод не изменяет исходные массивы.

Синтаксис:

```
имя_массива1.concat(имя_массива2)
```

Пример 24. Использование метода `concat()` объекта `Array` (листинг 15, рисунок 7).

Листинг 15

```
<html>
<head>
<title>Массивы</title>
</head>
<body>
<script language="JavaScript">
<!--
myArray = new Array("first element",
"second element","third element")
qwe = new Array("а", "б", "в", "г")
a=myArray.length
document.write("длина массива myArray " + a+
               "<br>")
q=qwe.length
document.write("длина массива qwe " + q+ "<br>")
document.write("<b>объединяем массивы</b><br>")
s = qwe.concat(myArray)
b=s.length
document.write("длина нового массива s " + b+
               "<br> Новый массив:<br>")
for (var i = 0; i < b; i++)
{
document.write(s[i] + "<br>")
}
// -->
</script>
</body>
</html>
```

2. Метод `pop()` удаляет последний элемент массива и возвращает

его значение.

Синтаксис:
`имя_массива.pop()`

3. Метод `push()` добавляет к массиву указанное значение в качестве последнего элемента и возвращает новую длину массива.

Синтаксис:
`имя_массива.push(значение|объект)`

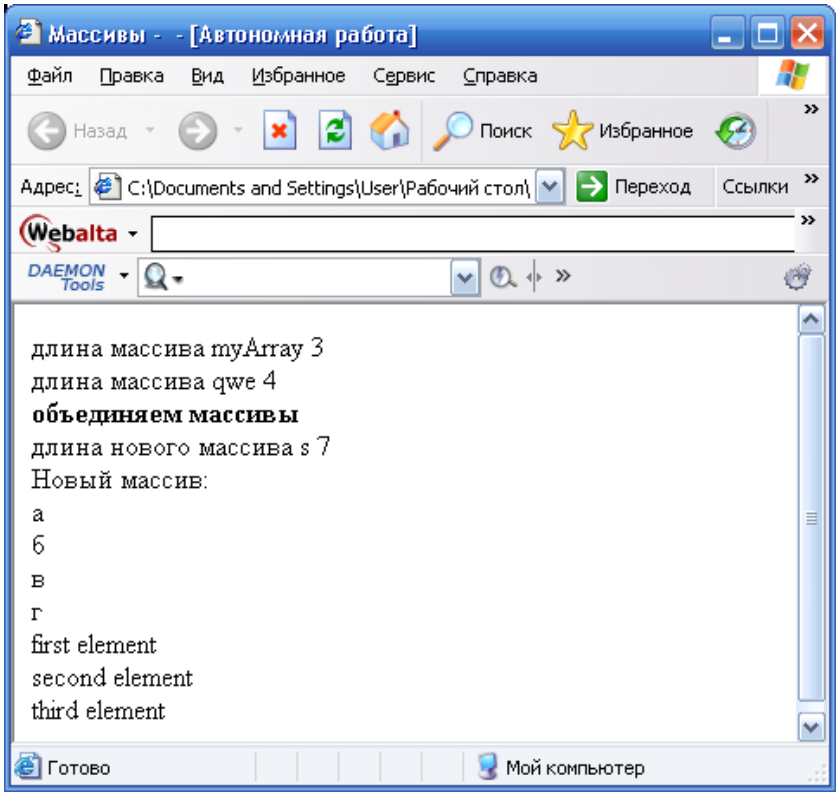


Рисунок 7 – **Использование метода `concat()` объекта `Array`**

4. Метод `join()` создает строку из элементов массива с указанным разделителем между ними, является строкой символов (возможно, пустой).

Синтаксис:
```
имя_массива.join(строка)
```

Пример 25. Использование метода `join()` объекта `Array` (листинг 16, рисунок 8).

Листинг 16

```
<html>
<head>
<title>Массивы</title>
</head>
<body>
<b>Исходные данные</b>
<br>
<script language="JavaScript">
<!--
myArray = new Array("first element",
"second element","third element")
for (var i = 0; i < 3; i++)
{
document.write(myArray[i] + "<br>")
}
s=myArray.join(", ")
document.write("<b>Результат</b><br>"+s)
// -->
</script>
</body>
</html>
```

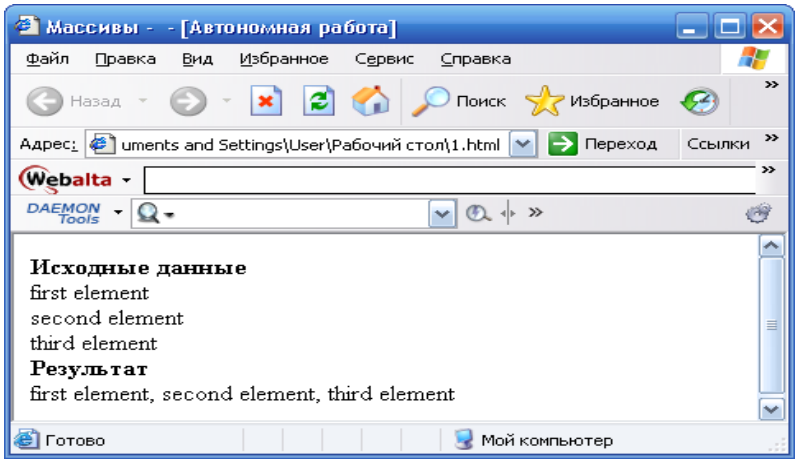


Рисунок 8 – **Использование метода `join()` объекта `Array`**

5. Метод `reverse()` изменяет порядок следования элементов массива на противоположный.

Синтаксис:

```
имя_массива.reverse()
```

Пример 26. Использование метода `reverse()` объекта `Array` (листинг 17, рисунок 9).

Листинг 17

```
<html>
<head>
<title>Массивы</title>
</head>
<body>
<b>Исходные данные</b>
<br>
<script language="JavaScript">
<!--
myArray = new Array("first element",
"second element","third element")
for (var i = 0; i < 3; i++)
{
document.write(myArray[i] + "<br>")
}
s=myArray.reverse()
```

```
document.write("<b>Результат</b><br>")
for (var i = 0; i < 3; i++)
{
document.write(myArray[i] + "<br>")
}
// -->
</script>
</body>
</html>
```

6. Метод `slise()` создает массив из элементов исходного массива с индексами указанного диапазона.

Синтаксис:

```
имя_массива.slise(индекс1[, индекс2])
```

Параметр `индекс2` не является обязательным. Если он не указан, то создаваемый массив содержит элементы исходного массива начиная с индекса `индекс1` и до конца. В противном случае создаваемый массив содержит элементы исходного массива начиная с индекса `индекс1` и до индекса `индекс2`, за исключением последнего.

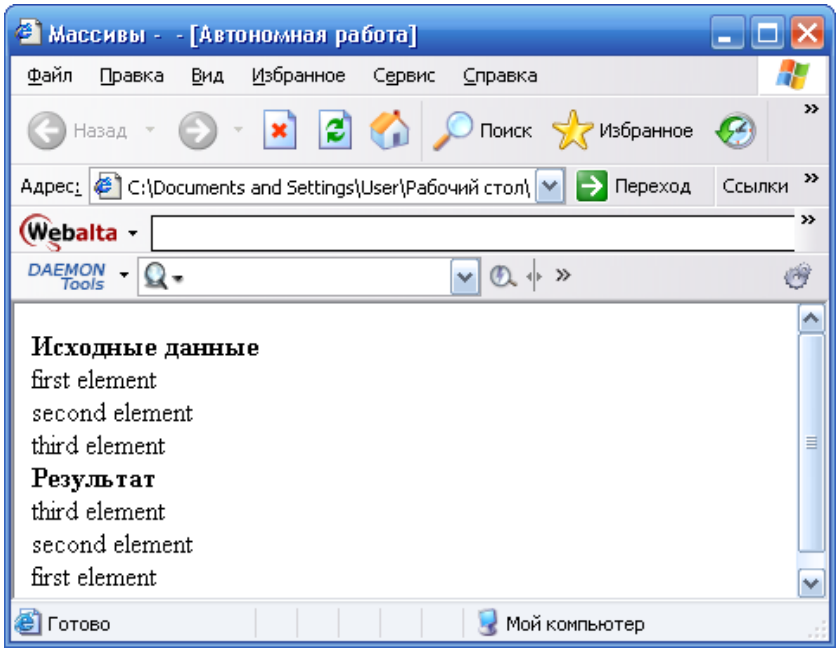


Рисунок 9 – **Использование метода `reverse()` объекта `Array`**

7. Метод `sort()` сортирует (упорядочивает) элементы массива с помощью функции сравнения.

Синтаксис:

```
имя_массива.sort([функция_сортировки])
```

Параметр `функция_сортировки` не обязателен. Если параметр не указан, то сортировка производится на основе сравнения ASCII-кодов символов значений. Это удобно для сравнения символьных строк, но не совсем подходит для сравнения чисел.

Пример 27. Использование метода `sort()` объекта `Array` для сортировки символьных элементов массива (листинг 18, рисунок 10).

Листинг 18

```
<html>
<head>
<title>Массивы</title>
</head>
<body>
```

```
<b>Исходные данные</b>
<br>
<script language="JavaScript">
<!--
myArray = new Array("s","e","d")
for (var i = 0; i < 3; i++)
{
document.write("myArray["+i+"]="+
               myArray[i] + "<br>")
}
s=myArray.sort()
document.write("<b>Результат</b><br>")
for (var i = 0; i < 3; i++)
{
document.write("myArray["+i+"]="+
               myArray[i] + "<br>")
}
// -->
</script>
</body>
</html>
```

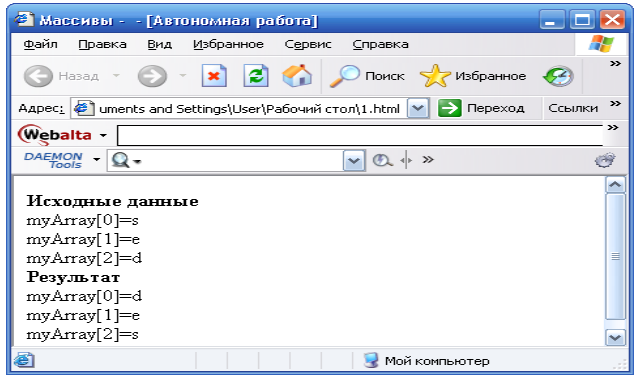


Рисунок 10 – **Использование метода `sort()` объекта `Array` для сортировки символьных элементов массива**

Можно создать свою собственную функцию для сравнения числовых элементов массива, с помощью которой метод `sort()` отсортирует весь массив. Имя этой функции передается методу в качестве параметра. При работе метода функции передаются два элемента массива, а ее код возвращает методу значение, указывающее, какой из элементов должен следовать за другим.

Пример 28. Использование метода `sort()` объекта `Array` для сортировки числовых элементов массива (листинг 19, рисунок 11).

Листинг 19

```
<html>
<head>
<title>Массивы</title>
</head>
<body>
<b>Исходные данные</b>
<br>
<script language="JavaScript">
<!--
myArray = new Array(1,6,9,9,3,5)
function g(myArray,qwe)
{
if(myArray > qwe) return 1
if(myArray < qwe) return -1
if(myArray == qwe) return 0
}
for (var i = 0; i < 6; i++)
{
document.write("myArray["+i+"]="+
               myArray[i] + "<br>")
}
qwe = myArray.sort(g)
document.write("<b>Результат</b><br>")
for (var i = 0; i < 6; i++)
{
document.write("qwe["+i+"]="+qwe[i] + "<br>")
}
// -->
</script>
</body>
</html>
```

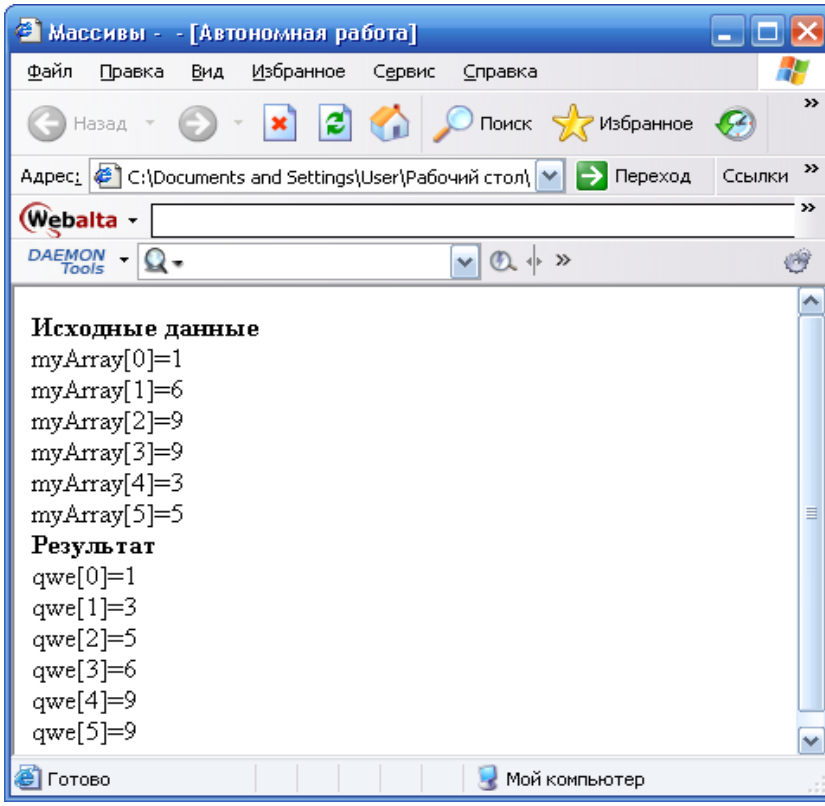


Рисунок 11 – **Использование метода `sort()` объекта *`Array`* для сортировки числовых элементов массива**

8. Метод `splise()` удаляет из массива несколько элементов и возвращает массив из удаленных элементов или заменяет значения элементов.

Синтаксис:

```
имя_массива.splise(индекс, количество[, элем1 [, элем2 [, ...элемN]]])
```

Первые два параметра обязательны, а следующие – нет. Первый параметр является индексом первого удаляемого элемента, а второй – количеством удаляемых элементов.

Метод `splice()` позволяет также заменить значения элементов исходного массива, если указаны третий и, возможно, последующие

параметры. Эти параметры представляют значения, которыми следует заменить исходные значения элементов массива. При таком использовании метода `splice()` важен параметр `индекс`, а второй `количество` может быть равным нулю. Если количество элементов замены больше значения второго параметра, то часть элементов исходного массива будет заменена, а часть элементов будет просто вставлена в него.

9. Метод `toLocaleString()` преобразует содержимое массива в символьную строку.

Синтаксис:

```
имя_массива.toLocaleString()
```

4.2. Объект `Date`

Во многих приложениях приходится отображать дату и время, подсчитывать количество дней, оставшихся до заданной даты, и т. п. Некоторые программы управляются с помощью значений дат и времени. В основе всех этих операций лежат текущие системные дата и время, установленные на компьютере.

В программе на языке JavaScript нельзя просто написать 30.10.2002, чтобы получить значение даты, с которым в дальнейшем можно производить операции. Значения даты и времени создаются как экземпляры объекта `Date`. При этом объект сам «знает», что не бывает 31 июня и 30 февраля, а в високосных годах 366 дней.

Объект `Date` содержит информацию о дате и времени. Он создается с использованием ключевого слова `new`.

Синтаксис:

```
имя_объекта_даты = new Date([параметры])
```

Аргумент `параметры` может принимать следующие значения:

- пустой параметр – в таком случае дата и время системные;
- строку, представляющую дату и время в виде «месяц, день, год, время», время представлено в 24-часовом формате;
- значения года, месяца, дня, часов, минут, секунд (например, строка «00,4,1,12,30,0» означает 1 апреля 2000 г., 12:30);
- целочисленные значения только для года, месяца и дня (например, «00,5,1» означает 1 мая 2000 г., сразу после полуночи, так как значения времени равны нулю).

Объект Date имеет множество методов, предназначенных для получения такой информации. Кроме того можно создавать и изменять объекты Date. Например, новую дату можно получать путем сложения или вычитания значений дат.

Методы объекта Date предназначены для действий с объектом Date.

Синтаксис:

```
имя_объекта_даты.метод()
```

Множество всех методов можно разделить на две категории: методы получения значений, названия которых имеют префикс get, и методы установки новых значений, названия которых имеют префикс set (таблица 6).

Таблица 6 – **Основные методы объекта Date**

Метод	Диапазон значений	Описание
getFullYear( )	**1970–...**	**Год**
getYear( )	**70–...**	**Год**
getMonth( )	**0–11**	**Месяц (январь = 0)**
getDate( )	**1–31**	**Число**
getDay( )	**0–6**	**День недели (воскресенье = 0)**
getHours( )	**0–23**	**Часы в 24-часовом формате**
getMinutes( )	**0–59**	**Минуты**
getSeconds( )	**0–59**	**Секунды**
setYear(знач)	**1970–...**	**Установка года (четырехзначного)**
setFullYear(знач)	**1970–...**	**Установка года**
setMonth(знач)	**0–11**	**Установка месяца (январь = 0)**
setDate(знач)	**1–31**	**Установка числа**
setDay(знач)	**0–6**	**Установка дня недели (воскресенье = 0)**
setHours(знач)	**0–23**	**Установка часов в 24-часовом формате**
setMinutes(знач)	**0–59**	**Установка минут**
setSeconds(знач)	**0–59**	**Установка секунд**

Пример 29. HTML-документ, в котором выводится текущее время (листинг 20, рисунок 12).

Листинг 20

```
<html>
<head>
<title>Дата и время</title>
</head>
<body>
<table cellpadding=5 width=100% border=0>
<tr>
<td width=95% bgcolor=gray align=left>
<font color=white>
Время:
<script language="JavaScript">
<!--
thedate=new Date()
d=thedate.getHours() +":"+thedate.getMinutes()
document.write(d)
// -->
</script>
</font>
</td></tr></table>
</body>
</html>
```

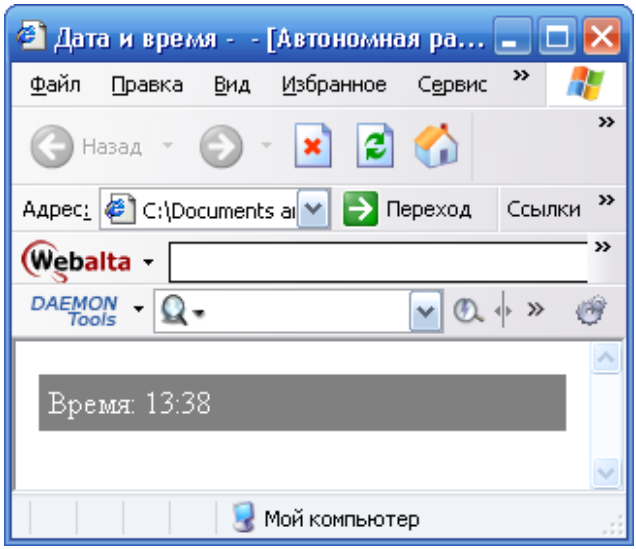


Рисунок 12 – **Вывод текущего времени**

4.3. Объект `Math`

Объект `Math` – встроенный в JavaScript объект, дающий доступ к константам и математическим функциям. Доступ к свойствам и методам объекта `Math` обеспечивается следующими выражениями:

```
Math.свойство
Math.метод(параметры)
```

Свойства объекта `Math` имеют в качестве своих значений математические константы (таблица 7).

Таблица 7 – **Свойства объекта `Math`**

Свойство	Описание
`E`	**Константа Эйлера. Приближенное значение 2.718**
`LN2`	**Значение натурального логарифма числа два. Приближенное значение 0.693**
`LN10`	**Значение натурального логарифма числа десять. Приближенное значение 2.302**
`LOG2_E`	**Логарифм `E` по основанию 2**
`LOG10_E`	**Десятичный логарифм `E`. Приближенное значение 0.434**

Свойство	Описание
`PI`	**Число пи. Приближенное значение 3.1415**
`SQRT2`	**Корень из двух**

Методы объекта `Math` представляют собой математические функции (таблица 8).

Таблица 8 – **Методы объекта `Math`**

Метод	Описание
`abs()`	**Возвращает абсолютное значение аргумента**
`acos()`	**Возвращает арккосинус аргумента**
`asin()`	**Возвращает арксинус аргумента**
`atan()`	**Возвращает арктангенс аргумента**
`ceil()`	**Возвращает большее целое число аргумента, округление в большую сторону. `Math.ceil(3.14)` вернет 4**
`cos()`	**Возвращает косинус аргумента**
`exp()`	**Возвращает экспоненту аргумента**
`floor()`	**Возвращает наибольшее целое число аргумента, отбрасывает десятичную часть**

Окончание таблицы 8

Метод	Описание
`log()`	**Возвращает натуральный логарифм аргумента**
`max()`	**Возвращает больший из 2 числовых аргументов. `Math.max(3,5)` вернет число 5**
`min()`	**Возвращает меньший из 2 числовых аргументов**
`pow()`	**Возвращает результат возведения в степень первого аргумента вторым. `Math.pow(5,3)` вернет 125**
`random()`	**Возвращает случайное число между 0 и 1**
`round()`	**Округление аргумента до ближайшего целого числа**
`sin()`	**Возвращает синус аргумента**
`sqrt()`	**Возвращает квадратный корень аргумента**
`tan()`	**Возвращает тангенс аргумента**

Пример 30. Генерация случайных чисел (листинг 21, рисунок 13).

Листинг 21

```
<html>
<head>
```

```
<title>Случайные числа</title>
</head>
<body>
<script language="JavaScript">
<!--
s= Math.random()
document.write("Случайное число от 0 до 1: "+s)
s= 5*Math.random()
document.write("<br>Случайное число от 0 до 5: "
                +s)
s= 5*Math.random()
d= Math.round(s)
document.write("<br>Целое случайное число от 0 до
                5: "+d)
s= 10 + (50 - 10)*Math.random()
d= Math.round(s)
document.write("<br>Целое случайное число от 10 до
                50: "+d)
// -->
</script>
</body>
</html>
```

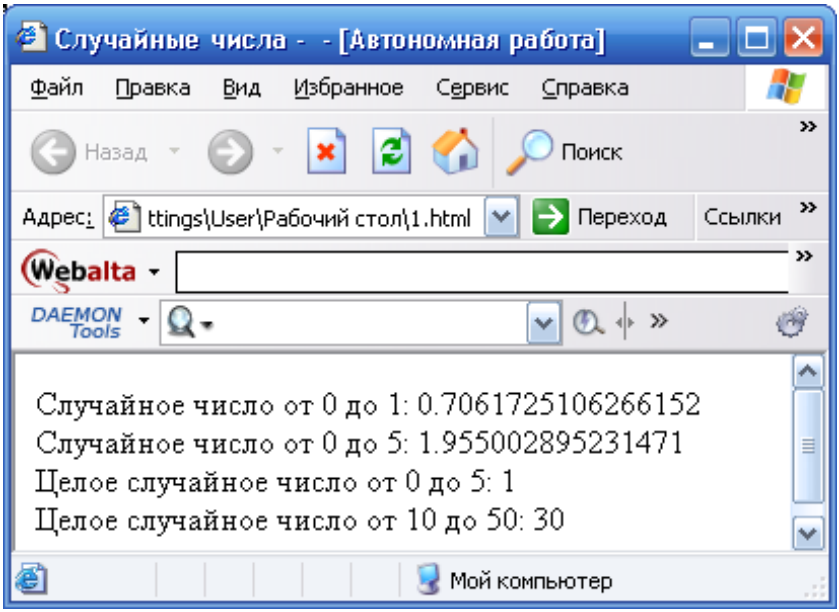


Рисунок 13 – **Генерация случайных чисел**

Пример 31. Применение случайных чисел для случайного выбора элемента массива (листинг 22).

Листинг 22

```
<html>
<head>
<title>Случайные числа</title>
</head>
<body>
<h2>Законы Мерфи</h2>
<script language="JavaScript">
<!--
myArray= new Array()
myArray[0]= "Соседняя очередь всегда движется
             быстрее"
myArray[1]= "Вероятность того, что бутерброд
            упадет маслом вниз, прямо
            пропорциональна стоимости ковра"
myArray[2]= "Зубная боль начинается в ночь на
```

```
            субботу"
myArray[3]= "Утерянное всегда находишь в последнем
            кармане"
myArray[4]= "Стоит запечатать письмо, как в голову
            приходят свежие мысли"
s= 4*Math.random()/*генерируем случайное число от
                    0 до 4*/
d= Math.round(s)/*округляем случайное число до
                  целого*/
document.write(myArray[d])
// -->
</script>
</body>
</html>
```

4.4. Объект `String`

Объект `String` используется для обработки строк. Например, когда требуется найти позицию вхождения одной строки в другую, вырезать из строки некоторую ее часть, разбить строку на отдельные элементы и создать из них массив и т. д.

Объект `String` создается с использованием ключевого слова `new`.

Синтаксис:

```
имя_объекта = new String("строковое_значение")
```

Здесь `имя_объекта` выполняет роль ссылки на строковый объект. Например, выражение

```
mystring = new String ("Привет!")
```

создает строковый объект `mystring` со значением «Привет!».

Можно создать строковый объект с помощью обычного оператора присвоения

```
имя_переменной = "строковое_значение"
```

или

```
var имя_переменной = "строковое_значение".
```

Доступ к свойствам и методам строкового объекта обеспечивается следующими выражениями:

```
имя_объекта.свойство
String.свойство
имя_объекта.метод([параметры])
String.метод([параметры])
```

У объекта `String` есть свойство `length`, которое возвращает длину строки, т. е. количество символов в строке.

Пример 32. Определение длины строки:

```
s="internet"
a= s.length // вернет значение 8
```

Основные методы объекта `String`

1. Метод `charAt()` возвращает символ, занимающий в строке указанную позицию. Индекс является числом.

Синтаксис:
```
строка.charAt(индекс)
```

Пример 33. Использование метода `charAt()`:

```
s="Привет".charAt(2) // значение равно "и"
```

2. Метод `charCodeAt()` преобразует символ в указанной позиции строки в его числовой эквивалент (код).

Синтаксис:
```
строка.charCodeAt([индекс])
```

3. Метод `fromCharCode()` возвращает строку символов, числовые коды которой указаны в качестве параметров.

Синтаксис:
```
String.fromCharCode(номер1 [, номер2 [, ... но-
мерN]])
```

4. Метод `concat()` производит конкатенацию строк – действует так же, как и оператор сложения «+» для строк, т. е. к строке `строка1` приписывается справа `строка2`.

Синтаксис:
```
строка1.concat(строка2)
```

Пример 34. Использование метода `concat()`:

```
s="Иван".concat(" Иванов")/* значение равно "Иван
                             Иванов" */
x = "Федор"
g=x.concat(" Иванов")/* значение равно "Федор
                        Иванов" */
```

5. Метод `indexOf()` производит поиск строки, указанной параметром, и возвращает индекс ее первого вхождения.

Синтаксис:
```
строка1.indexOf(строка2 [, индекс])
```

Метод производит поиск позиции первого вхождения `строки2` в строку `строка1`. Если поиск не удачен, то возвращается –1. Поиск в пустой строке всегда возвращает –1. Поиск пустой строки всегда возвращает 0. Второй параметр указывает индекс, с которого следует начать поиск.

Пример 35. Использование метода `indexOf()`:

```
m= "Соседняя очередь всегда движется быстрее"
e=m.indexOf("очередь")// 9
```

6. Метод `lastIndexOf()` производит поиск строки `строка2` и возвращает индекс ее первого вхождения в `строку1`, при этом поиск начинается с конца исходной строки, но возвращаемый индекс отсчитывается от ее начала, т. е. от 0. Метод аналогичен `indexOf()` и отличается лишь направлением поиска.

Синтаксис:
```
строка1.lastIndexOf(строка2 [, индекс])
```

Пример 36. Использование метода `lastIndexOf()`:

```
m= "Соседняя очередь всегда движется быстрее"
e=m.lastIndexOf("очередь")// 9
```

7. Метод `localeCompare()` позволяет сравнивать строки в кодировке Unicode, т. е. с учетом используемого браузером языка общения с пользователем.

Синтаксис:
```
строка1.localeCompare(строка2)
```

Если сравниваемые строки одинаковы, метод возвращает 0. Если `строка1` меньше чем `строка2`, то возвращается отрицательное число, в противном случае – положительное.

8. Метод `slice()` возвращает подстроку исходной строки, начальный и конечный индексы которой указываются параметрами, за исключением последнего символа.

Синтаксис:
```
строка.slice(индекс1 [, индекс2])
```

Если второй параметр не указан, то возвращается подстрока с начальной позицией `индекс1` и до конца строки. Отсчет позиций начинается с начала строки. Если второй параметр указан, то возвращается подстрока исходной строки начиная с позиции `индекс1` и до позиции `индекс2`, исключая последний символ. Если второй параметр отрицателен, то отсчет конечного индекса производится от конца строки.

Пример 37. Использование метода `slice()`:

```
m= "Соседняя очередь всегда движется быстрее"
e=m.slice(4,12)// возвращает «дняя оче»
```

9. Метод `split()` возвращает массив элементов, полученных из исходной строки.

Синтаксис:
```
строка.split(разделитель [, ограничитель])
```

Параметр `разделитель` является строкой символов, используемой в качестве разделителя строки на элементы. Параметр `ограничитель` – число, указывающее, сколько элементов строки, полученной при разделении, следует включить в возвращаемый массив. Если `разделитель` – пустая строка, то возвращается массив символов строки.

10. Метод `substr()` возвращает подстроку исходной строки, начальный индекс и длина которой указываются параметрами.

Синтаксис:
```
строка.substr(индекс[, длина])
```

Если второй параметр не указан, то возвращается подстрока с начальной позицией `индекс` и до конца строки. Отсчет позиций начинается с начала строки.

Пример 38. Использование метода `substr()`:

```
x = "mumu@gerasim.ru"
i = x.indexOf("@") // значение равно 4
n = x.substr(0, i) // значение равно "mumu"
d = x.substr(i+1) // значение равно "gerasim.ru"
```

11. Метод `substring()` возвращает подстроку исходной строки, начальный и конечный индексы которой указываются параметрами. Данный метод не изменяет исходную строку.

Синтаксис:
```
строка.substring(индекс1, индекс2)
```

Порядок индексов не важен, так как наименьший из них считается начальным. Символ, соответствующий конечному индексу, не включается в возвращаемую строку.

12. Метод `toLocaleLowerCase()` переводит строку в нижний регистр.

Синтаксис:
```
строка.toLocaleLowerCase()
```
Приведение строк к одному и тому же регистру требуется, например, при сравнении содержимого строк без учета регистра.

13. Метод `toLocaleUpperCase()` переводит строку в верхний

регистр. Приведение строк к одному и тому же регистру требуется, например, при сравнении содержимого строк без учета регистра.

Синтаксис:
```
строка.toLocaleUpperCase()
```

Тексты на Web-страницах обычно создаются и форматируются с помощью тегов HTML. Тоже самое можно создавать и с помощью сценариев. Например, чтобы вывести на Web-страницу строку «Привет всем!» полужирным шрифтом, в HTML-коде следует написать

```
<B>Привет всем!</B>
```

Чтобы подготовить эту же строку в таком же формате средствами JavaScript, в сценарии следует написать

```
"Привет всем!".bold()
```

Для форматирования строк здесь использован метод `bold()` объекта `String`. Выполнение этого выражения лишь создает отформатированную строку, но не выводит ее в окно браузера. Поэтому следует еще применить метод `write()` объекта `document` для записи этой строки в HTML-документ.

Пример 39. Код HTML-документа (листинг 23, рисунок 14).

Листинг 23
```
<html>
<script language="JavaScript">
<!--
s = "Привет всем!".bold()
document.write(s)
// -->
</script>
<p>Приветствие было вставлено сценарием JavaS-
cript</p>
</html>
```

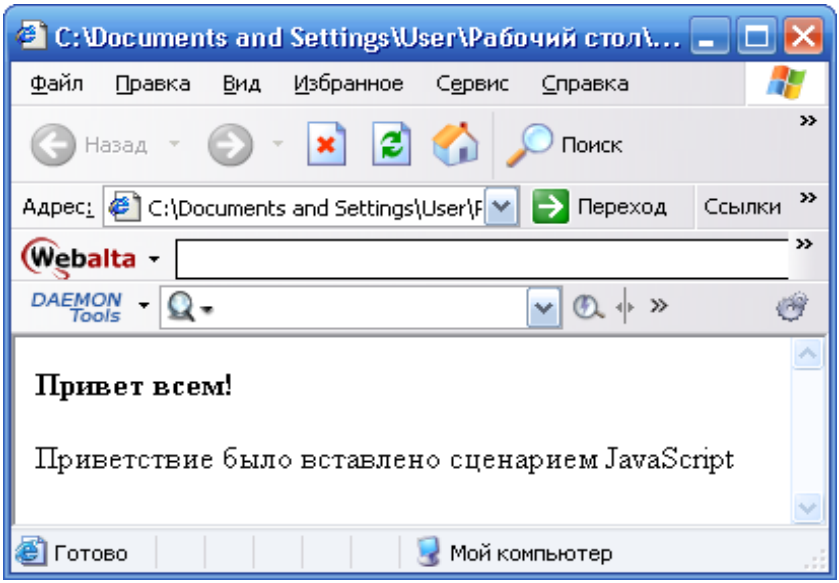


Рисунок 14 – **Результат выполнения листинга 23**

Методы форматирования строк носят названия, соответствующие тегам HTML (таблица 9). Их следует записывать только строчными буквами. Большинство методов форматирования не имеют параметров.

Синтаксис:

```
строка.метод([параметр])
```

Таблица 9 – **Методы форматирования строк объекта `String`**

Метод	Описание
`anchor("anchor_имя")`	Создает HTML якорь, который используется как гипертекстовая ссылка
`blink()`	Отображает мигающую строку, как если бы она находилась в теге `<BLINK>`
`bold()`	Отображает строку жирным шрифтом, как если бы она находилась в теге `<B>`
`fontcolor(значение цвета)`	Отображает строку с установленным цветом шрифта, как если бы она находилась в теге `<font color=color>`
`fontsize(число от 1 до 7)`	Отображает строку установленным размером шрифта, как если бы она находилась в теге `<font size=size>`

Окончание таблицы 9

Метод	Описание
italics()	**Отображает строку, отображаемую курсивом, как если бы она находилась в теге <i>**
link(расположение или URL)	**Создает гипертекстовую ссылку, по которой можно перейти на другой URL**
big()	**Отображает строку большим шрифтом, как если бы она находилась в теге <big>**
small()	**Отображает строку маленьким шрифтом, как если бы она находилась в теге <small>**
sub()	**Отображает строку как нижний индекс, как если бы она находилась в теге <sub>**
sup()	**Отображает строку как верхний индекс, как если бы она находилась в теге <sup>**

Пример 40. Использование метода `link()` объекта `String`:

```
"Сайт БТЭУ".link("i-bteu.by")
/* эквивалентно HTML-коду:
<a href = "i-bteu.by">Сайт БТЭУ</a> */
```

Задания для самостоятельного выполнения

1. Выполните скрипты, приведенные в листингах 12–23.
2. Найдите максимальный элемент массива из 5 элементов.
3. Введите текст в окно, создаваемое методом `prompt()`. В другое окно, создаваемое методом `prompt()`, введите букву. Определите позицию первого вхождения буквы в тексте и выведите на экран строку, начиная с этого символа, длиной 3 символа.
4. Напишите скрипт, показывающий текущую дату.
5. Напишите скрипт, который меняет цвет фона страницы и цвет основного текста в зависимости от времени суток. Днем фон белый, текст черный, а ночью – наоборот.

Лабораторная работа 5
ОСНОВЫ ОБЪЕКТНО-ОРИЕНТИРОВАННОГО ПРОГРАММИРОВАНИЯ В JAVASCRIPT

5.1. События JavaScript

Одним из главных назначений сценариев в HTML-документе является обработка событий, таких как щелчок кнопкой мыши на элементе документа, помещение указателя мыши на элемент, перемещение указателя с элемента, нажатие клавиши и т. п.

Большинство тегов HTML имеют специальные атрибуты, определяющие события, на которые могут отреагировать соответствующие элементы. Список всех допустимых событий обширный и рассчитан практически на все случаи жизни.

События называются довольно просто, например, щелчок левой кнопкой мыши – `onClick`; изменение в поле ввода данных – `onChange`; когда указатель мыши помещается на элемент HTML-документа – `onMouseOver`. Для одного и того же элемента можно определить несколько событий, на которые он будет реагировать.

Пример 41. Программа обработки события `onClick` (листинг 24).

Листинг 24

```
<html>
<head><title>События</title>
</head>
<body>
<form>
<input type="button" value="Щелкни!"
onClick="alert('Привет!')">
</form >
</body>
</html>
```

Пример 42. Программа обработки события `onClick` с использованием функции (листинг 25).

Листинг 25

```
<html>
<head>
<title>События</title>
<script language="JavaScript">
```

```
<!--
function Hello()
{
alert('Привет!')
}
// -->
</script>
</head>
<body>
<form>
<input type="button" value="Щелкни!"
onClick="Hello()">
</form>
</body>
</html>
```

Основные события, использующиеся в JavaScript

1. Событие `onClick` вызывается после щелчка левой кнопкой мыши на объекте. Например:

```
<input type="elementType" onClick="function">
```

2. Событие `onMouseOver` происходит, когда указатель мыши помещается над гиперссылкой. Например:

```
<a href="URL" onMouseOver="function">linkText</a>
```

3. Событие `onFocus` происходит в тот момент, когда пользователь переходит к элементу формы `select, text` или `textarea` для ввода данных. Например:

```
<input type="inputType" onFocus="function">
```

4. Событие `onBlur` происходит в тот момент, когда элемент формы `select, text` или `textarea` теряет фокус. Например:

```
<input type="elementType" onBlur="function">
```

5. Обработчик события `onSelect` вызывается в тот момент, когда выделен текст внутри элемента формы. Например:

```
<input type="textType" onSelect="function">
```

6. Событие onSubmit происходит в момент щелчка мышью на кнопке *Submit* для посылки данных формы на сервер. Например:

```
<tag onSubmit="function">
```

7. Событие onLoad вызывается, когда загрузка документа в окно или фрейм закончена. Например:

```
<body onLoad="function">
```
или
```
<frameset onLoad="function">
```

8. Событие onUnload вызывается, когда пользователь выходит из документа. Например:

```
<body onUnload="function">
```
или
```
<frameset onUnload="function">
```

9. Событие onChange происходит в тот момент, когда значение элемента формы select, text или textarea изменилось и элемент потерял фокус. Например:

```
<input type="elementType" onChange="function">
```

Пример 43. Обработка событий onMouseOver, onFocus, onBlur, onChange, onClick (листинг 26, рисунок 15).

Листинг 26

```
<html>
<head>
<title>События</title>
</head>
<body>
<h2 align=center>Обработка событий</h2>
<h4>
Событие onMouseOver - происходит, когда указатель
мыши помещается над гиперссылкой
</h4>
<a href="#" onMouseOver="alert('Бесплатный хостинг
где-то рядом')">Ссылка</a>
<form>
```

```
<h4>
Событие onFocus - происходит в тот момент, когда
пользователь переходит к элементу формы select,
text или textarea для ввода данных
</h4>
<input type="text" size="45"
value="Щелкни в поле и посмотри на строку
состояния" onFocus="window.status='Текст в строке
состояния';">
<h4>
Событие onBlur - происходит в тот момент, когда
элемент формы select, text или textarea теряет фокус
</h4>
<input type=" text" size="45"
value="Впишите свое имя и щелкните по другой
строке" onBlur=" alert('Вы изменили ответ –
уверены, что он правильный?');">
<h4>
Событие onChange - происходит в тот момент, когда
значение элемента формы select, text или textarea
изменилось и элемент потерял фокус
</h4>
<input type=" text"
size="45"
value="Измените текст и щелкните по другой строке"
onChange=" window.status='Текст был изменен';">
<h4>
Событие onClick - происходит после щелчка левой
кнопкой мыши на объекте
</h4>
<input type="reset" value="Щелкни!"
onClick="alert('Вы щелкнули по кнопке сброса
данных!')">
</form>
</body></html>
```

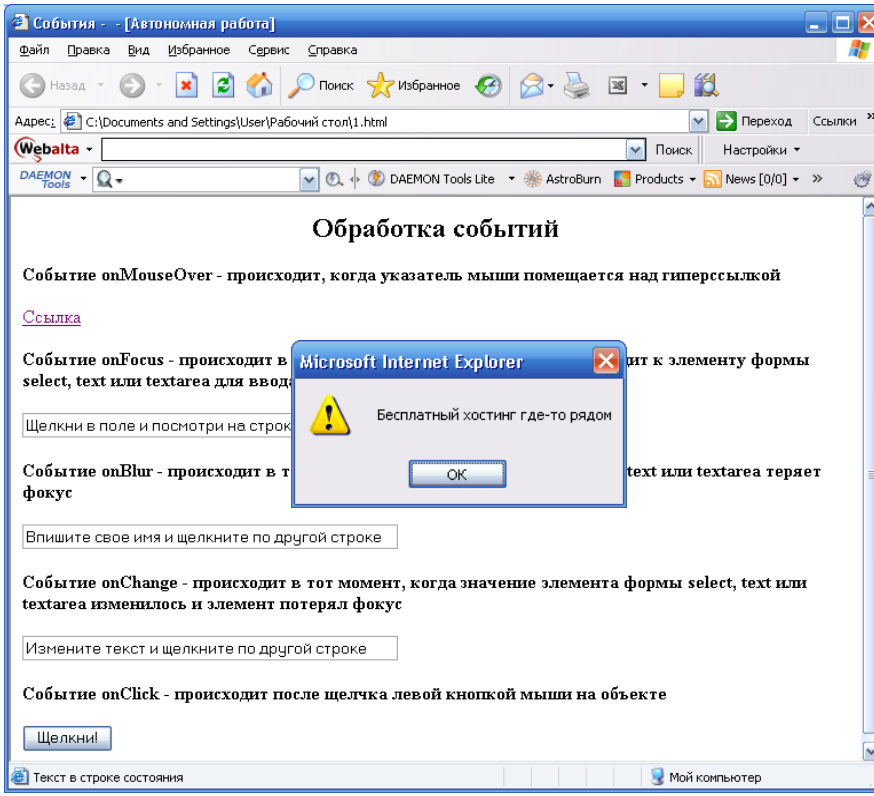


Рисунок 15 – **Результат выполнения листинга 26**

Примечание – Окно сообщения появилось в результате обработки события `onMouseOver`, текст в строке состояния изменился в результате обработки события `onFocus`.

5.2. Объекты JavaScript

В языке JavaScript имеется три типа объектов:

- встроенные объекты;
- объекты браузера;
- объекты, которые программист создает самостоятельно.

С точки зрения сценария JavaScript, браузер представляется иерархически организованным набором объектов (см. рисунок 1). Объекты браузера являются тем интерфейсом, с помощью которого сценарий

JavaScript взаимодействует с пользователем и документом HTML, загруженным в окно браузера, а также с самим браузером.

5.2.1. Объект `window`

Объект `window` обычно соответствует главному окну браузера и является объектом верхнего уровня в языке JavaScript, так как документы открываются в окне.

Основные свойства объекта `window`

1. Свойство `defaultStatus` выводит по умолчанию сообщение в строке состояния в нижней части окна браузера.

Синтаксис:
```
[windowName.]defaultStatus = "строка"
```

2. Свойство `status` устанавливает текст основного или временного сообщения в строке состояния.

Синтаксис:
```
[windowName.]status = "строка"
```

3. Свойство `frames` содержит массив фреймов, содержащихся в окне.

Синтаксис:
```
[windowName.][parent.]frameName, [win-
dowName.][parent.]frames[index]
```

4. Свойство `name` задает заголовок окна.

Синтаксис:
```
windowRef.name = "строка"
```

5. Свойство `window` является синонимом текущего окна.

Синтаксис:
```
[windowName.]window
```

Основные методы объекта `window`

1. Метод `alert()` выводит на экран диалоговое окно с кнопкой *OK* и сообщением.

Синтаксис:
```
[window.]alert("AlertMessage")
```

2. Метод `open()` создает новый экземпляр окна.

Синтаксис:
```
[window.]open("URL","windowName"
[,"windowFeatures"])
```

3. Метод `close()` закрывает текущий экземпляр окна.

Синтаксис:
```
[window.]close()
```

4. Метод `prompt()` выводит на экран диалоговое окно для ввода данных.

Синтаксис:
```
[windowName.]prompt(message [inputDefault])
```

5. Метод `confirm()` выводит на экран окно сообщения с кнопками *Yes* и *No* и возвращает значение `true` или `false` в зависимости от нажатой кнопки.

Синтаксис:
```
[window.]confirm(ConfirmMessage)
```

6. Метод `setTimeout()` исполняет выражение по истечении указанного в миллисекундах промежутка времени.

Синтаксис:
```
[window.]setTimeout(timerID)
```

Пример 44. Использование метода `close()` объекта `window` (листинг 27).

Листинг 27
```
<html>
```

```
<head>
<title>Объекты</title>
</head>
<body>
<script language="JavaScript">
<!--
if (confirm("Уверены, что хотите просмотреть
 документ?") )
{
alert("Вы выбрали просмотр документа")
}
else
{
alert("Вы отказались от просмотра документа")
window.close()
}
// -->
</script>
</body>
</html>
```

Как известно, HTML-документ загружается в окно браузера. Можно открыть несколько таких окон и загрузить в них различные документы, а также разбить одно окно на несколько прямоугольных областей, называемых фреймами.

Пример 45. Загрузка документа с именем `2.html` в новом окне (листинги 28 и 29, рисунок 16).

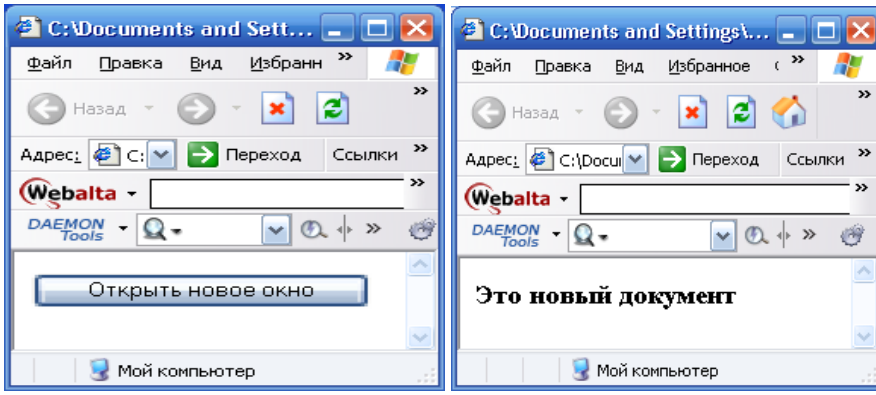
Листинг 28

```
<html>
<head>
<title>Объекты</title>
<script language="JavaScript">
<!--
function openWin()
{
window.open("2.html")
}
// -->
</script>
</head>
```

```
<body>
<form>
<input type="button" value="Открыть новое окно"
onClick="openWin()">
</form>
</body>
</html>
```

Листинг 29

```
<html>
<head>
<title>Объекты</title>
</head>
<body>
<h3>Это новый документ</h3>
</body>
</html>
```



а *б*

Рисунок 16 – **Результат выполнения листингов 28 и 29: исходное окно (*а*) и новое окно (*б*)**

Можно управлять самим процессом создания окна, например, указать, должно ли новое окно иметь строку статуса, панель инструментов или меню. Кроме того можно задать размер окна (таблица 10).

Таблица 10 – **Основные свойства нового экземпляра окна**

Атрибут	Значение	Описание
`height`	`pixelheight`	**Определяет высоту окна в пикселях**

Окончание таблицы 10

Атрибут	Значение	Описание
location	**[yes|no] | [1|0]**	**Определяет наличие поля location**
menubar	**[yes|no] | [1|0]**	**Определяет наличие меню**
resizable	**[yes|no] | [1|0]**	**Определяет наличие рамки окна, позволяющей менять размеры окна**
scrollbars	**[yes|no] | [1|0]**	**Определяет наличие линеек прокрутки**
status	**[yes|no] | [1|0]**	**Определяет наличие строки состояния**
toolbar	**[yes|no] | [1|0]**	**Определяет наличие панели инструментов**
width	**pixelwidth**	**Определяет ширину окна в пикселях**

По умолчанию атрибутам всегда присваивается значение `yes`, а размер нового окна (если он не задан) соответствует размеру предыдущего. Атрибуты можно указывать в произвольном порядке (листинг 30, рисунок 17).

Пример 46. Загрузка документа с именем `2.html` в новом окне с именем `displayWindow` шириной 400 пикселей, высотой 300 пикселей, без строки состояния, панели инструментов и меню (листинг 30, рисунок 17).

Листинг 30

```
<head>
<title>Объекты</title>
<script language="JavaScript">
<!--
function openWin()
{
window.open("2.html", "displayWindow", "width=400,
height=300, status=no, toolbar=no, menubar=no")
}
// -->
</script>
</head>
<body>
<form>
<input type="button" value="Открыть новое окно"
onClick="openWin()">
```

```
</form>
</body>
</html>
```

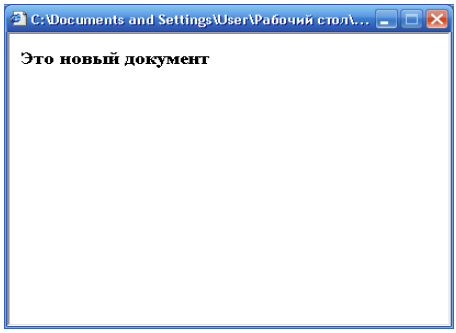


Рисунок 17 – **Результат выполнения листинга 30**

У объекта `window` имеется синоним `self`, используемый при обращении к окну, содержащему текущий документ. Идентификатор `self` применяется в многооконных или многофреймовых системах, когда требуется указать окно с документом, в котором находится данный сценарий. Его рекомендуется вставлять, чтобы не запутаться. При запуске сценария в ссылках на объекты только текущего документа идентификаторы `window` и `self` можно опускать.

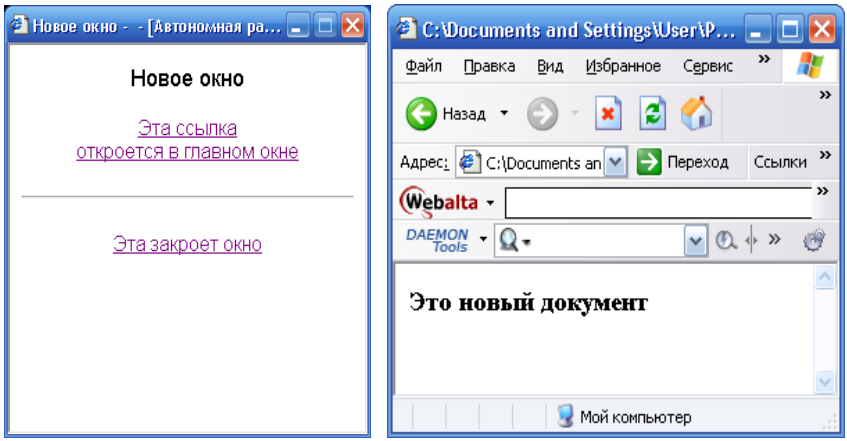
Рассмотрим динамическое создание документов, т. е. написание скрипта таким образом, чтобы можно было создавать новые HTML-страницы.

Пример 47. Создание нового окна (листинг 31, рисунок 18).

Листинг 31

```
<html>
<head>
<title>Объекты</title>
<script language="JavaScript">
<!--
function openw()
{
var win=window.open("", "newwin", "height=300,
width=300")
win.document.write("<html>")
win.document.write("<title>Новое окно</title>")
```

```
win.document.write("<body bgcolor='white'>")
win.document.write("<center>")
win.document.write("<font size=+1>Новое окно
                    </font><p>")
win.document.write("<a href='2.html'
                    target='main window'>
                    Эта ссылка<br>откроется в
                    главном окне</a><p>")
/* target указывает имя окна, в которое будет по-
мещен ответ */
win.document.write("<p><hr><p>")
win.document.write("<a href=''
                    onClick = 'self.close()'>
                    Эта закроет окно</a><p>")
win.document.write("</center></html>")
self.name="main window"//устанавливает имя окна
}
// -->
</script>
</head>
<body onLoad="openw()">
</body>
</html>
```



а *б*

Рисунок 18 – **Результат выполнения листинга 31: новое окно (*а*) и открытая ссылка в исходном окне (*б*)**

5.2.2. Объект document

Объект document создается браузером во время загрузки страницы. Объект document является одним из основных в JavaScript и содержит информацию о текущем документе – заголовок, цвет фона, имеющиеся в документе формы и т. п. Эти свойства определяются в теге <body>…</body>.

Основные свойства объекта document

1. Свойство action – отображение атрибута action тега <form> – возвращает строку, состоящую из URL назначения для данных, введенных в форму. Это значение может быть установлено или изменено как до, так и после загрузки и форматирования документа.

Синтаксис:
```
document.formName.action
document.forms[index].action
```

2. Свойство bgColor определяет фоновый цвет документа.

Синтаксис:
```
document.bgColor
```

3. Свойство fgColor устанавливает цвет текста, выводимого на странице.

Синтаксис:
```
document.fgColor
```

4. Свойство lastModified содержит строку только для чтения, хранящую дату последнего изменения текущего документа.

Синтаксис:
```
document.lastModified
```

5. Свойство location возвращает строку с URL текущего документа.

Синтаксис:

document.location

6. Свойство referer возвращает URL документа, который привел к текущему документу.

Синтаксис:
document.referer

7. Свойство title определяет значение только для чтения, указанное внутри тега <title>.

Синтаксис:
document.title

Коллекции объекта document

1. Коллекция forms содержит массив объектов, соответствующих формам, созданных в тегах HTML в том же порядке.

Синтаксис:
document.forms

2. Коллекция links содержит массив гипертекстовых связей.

Синтаксис:
document.links[index]

3. Коллекция images содержит массив изображений, ссылки на которые заданы в текущем документе.

Синтаксис:
document.images[index]

Основные методы объекта document

1. Метод open() создает новый документ. Метод открывает поток вывода для методов write или writeln. Если тип MIME является версией text или image (например, text/html или image/gif), документ будет открыт для показа.

Синтаксис:
document.open([MIMEtype])

2. Метод write() записывает строку или несколько строк в окно

документа.

Синтаксис:

```
document.write(string)
```

3. Метод `writeln()` записывает строку или несколько строк в окно документа и добавляет символ новой строки в конце вывода.

Синтаксис:

```
document.writeln(string)
```

4. Метод `close()` закрывает документ, посылает браузеру информацию об изменениях в документе.

Синтаксис:

```
document.close()
```

5. Метод `clear()` очищает текущий документ.

Синтаксис:

```
document.clear()
```

Пример 48. Использование свойств `bgColor`, `fgColor`, `location`, `lastModified` объекта `document` (листинг 32).

Листинг 32

```
<html>
<head>
<title>Объекты</title>
</head>
<body bgColor=yellow>
<script language="JavaScript">
<!--
var bgc = document.bgColor
var fgc = document.fgColor
var url = document.location
var lm = document.lastModified
document.write("Цвет фона этой страницы был
                установлен <b>"+bgc+"</b>")
document.write("<br>URL этой страницы<b>" +url+
                "</b>")
document.write("<br>Последние изменения внесены –
```

```
                    <b>"+lm+"</b>")
// -->
</script>
</body>
</html>
```

Пример 49. Изменение свойства `bgColor` объекта `document` (листинг 33).

Листинг 33

```
<html>
<head>
<title>Объекты</title>
</head>
<body bgColor=yellow>
<h3>Выберите цвет фона</h3>
<form>
<input type="button" value="Красный"
onClick="document.bgColor='red'">
<input type="button" value="Белый"
onClick="document.bgColor='white'">
<input type="button" value="Вернуть исходный"
onClick="document.bgColor='yellow'">
</form>
</body>
</html>
```

5.2.3. Объект `location`

Объект `location` сохраняет местоположение текущего документа в виде адреса этого документа. При управлении объектом `location` существует возможность изменять URL документа. Объект `location` связан с текущим объектом `window` – окном, в которое загружен документ.

Основные свойства объекта `location`

1. Свойство `hash` возвращает часть URL, начинающуюся с символа #, т. е. закладку.

Синтаксис:

```
document.linkName.hash
document.links[index].hash
document.location.hash
```

2. Свойство `hostname` возвращает или изменяет строку с именем домена или IP-адресом URL.

Синтаксис:
```
location.hostname
linkName.hostname
links[index].hostname
```

3. Свойство `href` возвращает строку, содержащую полный URL текущего документа.

Синтаксис:
```
location.href
linkName.href
links[index].href
```

4. Свойство `pathname` извлекает из URL часть, которая содержит путь.

Синтаксис:
```
location.pathname
link.pathname
links[index].pathname
```

5. Свойство `port` извлекает из URL номер порта компьютера.

Синтаксис:
```
location.port
link.port
links[index].port
```

6. Свойство `protocol` возвращает метод доступа к объекту (протокол передачи данных).

Синтаксис:
```
location.protocol
link.protocol
```

```
links[index].protocol
```

7. Свойство `target` возвращает строку, указывающую имя окна, в которое будет помещен ответ после посылки данных формы серверу.

Синтаксис:

```
location.target
link.target
links[index].target
```

Методов для объекта `location` не существует и с обработчиками событий он не связан.

Пример 50. Использование свойства `protocol` объекта `location` (листинг 34).

Листинг 34

```
<html>
<head>
<title>Объекты</title>
</head>
<body>
<script language="JavaScript">
<!--
document.write("Протокол передачи данных - " +
                location.protocol)
// -->
</script>
</body>
</html>
```

5.2.4. Объект `history`

Объект `history` содержит список URL-адресов, посещенных в текущем сеансе. Единственным свойством объекта является свойство `length`, которое определяет количество элементов в списке `history`.

Основные методы объекта `history`

1. Метод `back()` вызывает переход к предыдущему URL из списка

просмотренных документов в текущей сессии работы с браузером.

Синтаксис:

```
history.back()
```

2. Метод `forward()` загружает следующий документ из списка URL-адресов, просмотренных за текущий сеанс работы с браузером.

Синтаксис:

```
history.forward()
```

3. Метод `go()` загружает документ из списка страниц, посещенных за текущий сеанс работы браузера.

Синтаксис:

```
history.go(argument_Or_URL)
```

5.2.5. Объект `navigator`

Объект `navigator` содержит информацию о браузере на клиентском компьютере. Объект `navigator` возвращает информацию об имени и версии браузера. Одно из основных применений этого объекта состоит в определении платформы, используемой на клиентском компьютере для учета особенностей конкретного браузера, подобных обработке символа новой строки или генерации случайных чисел.

Основные свойства объекта `navigator`

1. Свойство `appCodeName` возвращает строку только для чтения с кодовым именем браузера.

Синтаксис:

```
navigator.appCodeName
```

2. Свойство `appName` возвращает строку только для чтения с именем браузера.

Синтаксис:

```
navigator.appName
```

3. Свойство `appVersion` возвращает строку с информацией о версии браузера.

Синтаксис:

```
navigator.appVersion
```

4. Свойство `userAgent` возвращает заголовок, посылаемый как часть протокола HTTP от клиента к серверу, для идентификации типа клиента.

Синтаксис:

```
navigator.userAgent
```

Методы и события не определены для этого объекта.

Пример 51. Использование свойств `appName`, `appVersion`, `appCodeName`, `userAgent` объекта `navigator` (листинг 35).

Листинг 35

```
<html>
<head>
<title>Объекты</title>
</head>
<body>
<script language="JavaScript">
<!--
var an = navigator.appName
var av = navigator.appVersion
var acn = navigator.appCodeName
var ua = navigator.userAgent
document.write("Вы пользуетесь <b>" +an+ "</b>,
                <br>версия " +av+ ".")
document.write("<br><br>Кодовое название " +acn+
               ", <br>заголовок "+ua+ "." )
// -->
</script>
</body>
</html>
```

Задания для самостоятельного выполнения

1. Выполните скрипты, приведенные в листингах 24–35.

2. Измените листинг 22 таким образом, чтобы посмотреть закон Мерфи можно было по щелчку на кнопке, а сам закон выводился в окне сообщения.

3. Напишите скрипт так, чтобы при щелчке по гиперссылке возникало окно сообщения.

4. Измените предыдущее задание так, чтобы при наведении курсора мыши на ссылку выполнялась процедура, выводящая в окно браузера фразу «Hello, world!».

5. Составьте документ так, чтобы окно сообщения для ввода информации предлагалось только после наведения курсора мыши на ссылку, и введенная пользователем текстовая строка выводилась в виде окна сообщения или в окно браузера.

6. Напишите скрипт, реализующий сложение двух чисел по щелчку на кнопке. Числа вводятся пользователем.

7. Напишите скрипт, содержащий функцию, которая бы открыла окно с зеленым фоном и приветствием «Привет, имя пользователя, вот твое окно!». Если пользователь не ввел имя, то новое окно не должно открываться. Создайте кнопку, которая закроет это окно (*).

8. Создайте форму со следующими элементами:

- поле ввода с просьбой ввести имя, при вводе которого в строке состояния должна появиться фраза «Впишите сюда свое имя»;
- два переключателя, позволяющие выбрать ответ на вопрос о том, что больше нравится пользователю – мороженое или шоколад, строка состояния должна отображать выбранный вариант ответа;
- кнопку отправки данных, при нажатии на которую должно появиться окно предупреждения, благодарящее пользователя за участие в опросе.

9. Напишите скрипт, который отображает диалоговое окно с кнопками *Ок* и *Отмена* и выводит в окно браузера сообщение о том, какая из кнопок была нажата (используйте конструкцию `if(confirm("Сообщение")) ...`).

10. Создайте документ с кнопкой, при нажатии на которую выводилось бы диалоговое окно, которое при нажатии на кнопку *OK* открывало второй документ и при нажатии на кнопку *Cancel* закрывало текущий документ.

Лабораторная работа 6
ИСПОЛЬЗОВАНИЕ ГРАФИЧЕСКИХ ИЗОБРАЖЕНИЙ

6.1. Объект `image`

При создании Web-страниц часто возникает необходимость разместить графическое изображение, заменить одно графическое изображение на другое, организовать слайд-шоу и т. д.

За размещение изображения в языке HTML отвечает тег <img>, для которого в качестве значения атрибута src устанавливается URL графического файла.

Синтаксис тега <img>:

```
<img src="fileOrUrl" [alt="altText"]
[lowsrc="fileOrUrl"] [width=w] [heigth=h]
[border=b] [vspace=v] [hspace=h]>
```

Примечание – В атрибуте src содержится URL файла, который нужно вывести в документе. Рисунок должен храниться в формате *GIF, JPEG* или *PNG*. С помощью атрибута alt задается альтернативный текст, появляющийся на экране в момент загрузки текста, если пользователь заблокировал вывод изображений, и поясняющая надпись под курсором мыши. Атрибут lowsrc позволяет предварительно выводить на экран изображение с низким разрешением. При этом рисунок загружается в два этапа. Атрибуты width и height позволяют задать размеры рисунка в пикселях, атрибут border – ширину рамки в пикселях, а атрибуты vspace и hspace – размеры вертикального и горизонтального зазоров между границами изображения и другими элементами документа.

В браузере рисунки рассматриваются как объекты image объекта document, а все рисунки, содержащиеся в текущем документе, помещаются в массив images, который можно использовать для обращения к любому рисунку, определяемому тегом <img>.

Суть использования графических изображений заключается в том, чтобы с помощью сценария JavaScript изменить значения атрибутов тега <img>.

Для обращения к свойствам объекта image используется следующий синтаксис:

- `document.images[i].propertyName`

Здесь i – индекс элемента массива, который соответствует нужному рисунку. Первым рисунком в документе будет document.images[0].

- `document.значение_name.propertyName`

Здесь значение_name – значение атрибута name тега <img>. В

таком случае не имеет значения, сколько изображений на странице и в каком порядке они располагаются, если каждой нужной картинке присвоено отдельное имя.

Массив `images` является свойством объекта `document`, поэтому при обращении к рисунку необходим префикс `document` к имени массива.

Все свойства объекта `image` соответствуют атрибутам тега `<img>`, за исключением свойства `complete`. Эти свойства, кроме свойств `src` и `lowsrc`, значения которых могут быть изменены динамически, имеют значения только для чтения (таблица 11).

Таблица 11 – **Свойства объекта `image`**

Свойство	Описание
`src`	**Соответствует атрибуту `src` тега `<img>`**
`lowsrc`	**Соответствует атрибуту `lowsrc` тега `<img>`**
`height`	**Соответствует атрибуту `height` тега `<img>`**
`width`	**Соответствует атрибуту `width` тега `<img>`**
`border`	**Соответствует атрибуту `border` тега `<img>`**
`vspace`	**Соответствует атрибуту `vspace` тега `<img>`**
`hspace`	**Соответствует атрибуту `hspace` тега `<img>`**
`complete`	**Содержит булевое значение, которое указывает, загружен рисунок в браузер или нет (`true` – загружен, `false` – нет)**
`type`	**Для объектов `image` содержит значение `"image"`**

Перед загрузкой рисунка появляется его рамка, внутри которой отображается строка, заданная в атрибуте `alt` (в IE5+ пользователь при желании может отключить рамки с `alt`-текстом, отображаемые в момент загрузки рисунка). Рисунок можно изменять динамически, присваивая атрибуту `src` или `lowsrc` в качестве значения новый URL.

6.2. Смена картинки через обработку событий

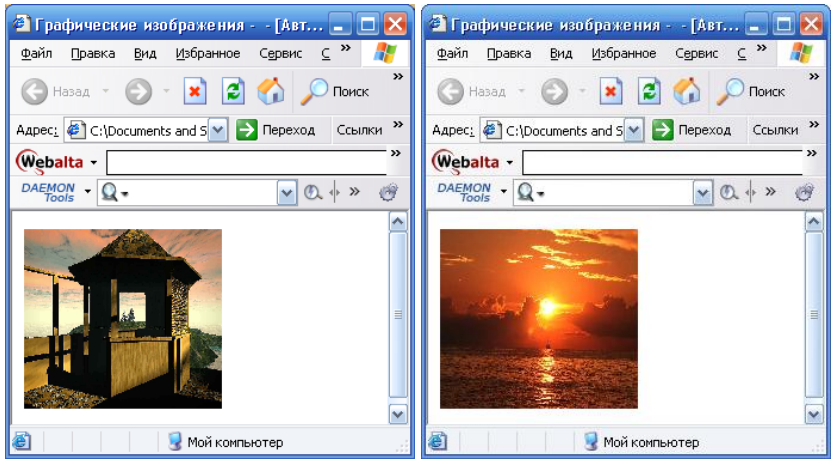
Для смены графического изображения можно использовать обработчики событий, например, `onMouseOver` и `onMouseOut`. Обработчики событий являются атрибутами тега `<a href>...</a>`. В этом случае не требуются тэги `<script>` и `</script>`.

Пример 52. Смена картинки при наведении указателя мыши на

картинку. Обработчик события `onMouseOver` меняет источник изображения на `1.jpg`, а обработчик события `OnMouseOut` заставляет объект изображение показывать `2.jpg` (листинг 36, рисунок 19).

Листинг 36

```
<html>
<head>
<title>Графические изображения</title>
</head>
<body>
<a href="#"
onMouseOver="document.pic.src='1.jpg'"
onMouseOut="document.pic.src='2.jpg'">
<img src="2.jpg" name="pic"
width=166 height=150 border=0>
</a>
</body>
</html>
```



а *б*

Рисунок 19 – **Результат выполнения листинга 36: исходный рисунок (*а*) и рисунок, появляющийся при наведении указателя мыши (*б*)**

6.3. Смена картинок через функцию

Обработчики событий могут быть использованы для вызова функции.

Для многократного повторения больше подходит функция, но для этого каждый раз нужно давать функции и изображению новое имя.

Пример 53. Смена картинки при наведении указателя мыши на картинку. Обработчик события `onMouseOver` вызывает функцию `up()`, а обработчик события `OnMouseOut` – функцию `down()` (листинг 37).

Листинг 37

```
<html>
<head>
<title>Графические изображения</title>
<script language="JavaScript">
<!--
function up()
{
document.pic.src="1.jpg"
}
function down()
{
document.pic.src="2.jpg"
}
// -->
</script>
</head>
<body>
<center>
<h2>Пример анимации</h2>
<a href="#"
onMouseOver="up()" onMouseOut="down()">
<img SRC="2.jpg" name="pic" border=0 width=260
height=250 >
</a>
</body>
</html>
```

Эффект при просмотре тот же, что и в предыдущем примере.

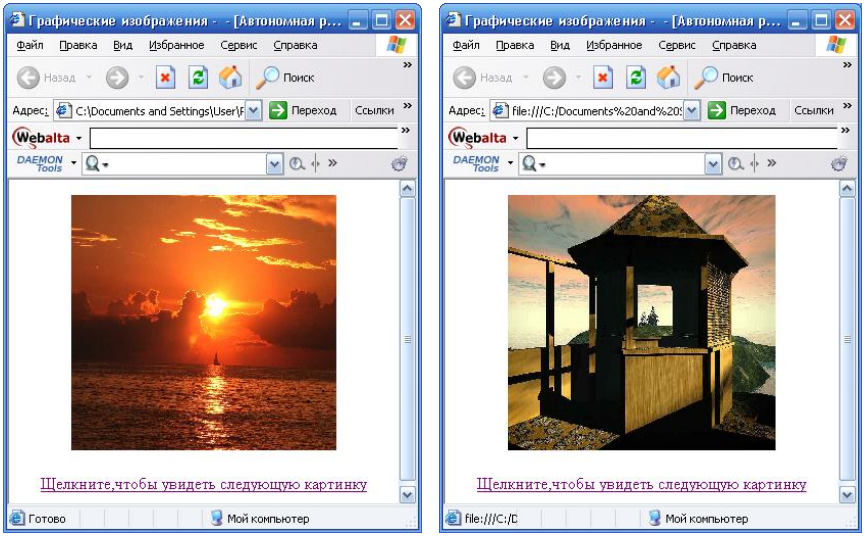
6.4. Слайд-шоу

Организовать слайд-шоу можно с использованием массивов. В таком случае каждый элемент массива – это URL графического изображения.

Пример 54. Создание слайд-шоу (листинг 38, рисунок 20).

Листинг 38

```
<html>
<head>
<title>Графические изображения</title>
<script language="JavaScript">
<!--
img = new Array()
img[0] = "1.jpg"
img[1] = "2.jpg"
img[2] = "3.jpg"
var num=0
function slideshow()
{
num=num+1
if (num==3)
{
num=0
}
document.animal.src=img[num]
}
//-->
</script>
</head>
<body>
<center>
<img src="1.jpg" name="animal" border=0 width=260
height=250 >
<p>
<a href="#" onClick="slideshow()">
Щелкните,чтобы увидеть следующую картинку</a>
</center>
</body>
</html>
```



а *б*

Рисунок 20 – **Результат выполнения листинга 38: первый рисунок (*а*) и второй рисунок (*б*), появляющийся после щелчка на гиперссылке**

Задания для самостоятельного выполнения

1. Выполните скрипты, приведенные в листингах 36–38.
2. Создайте навигационную панель, состоящую из изображений, меняющихся при наведении на них мышью. В скрипте не используйте теги `<script language = "JavaScript"> … </script>`.
3. Переделайте задание 2 таким образом, чтобы скрипт работал с использованием функций.
4. Напишите скрипт, в котором смена одной картинки на другую происходит при наведении указателя мыши на текст под картинкой.
5. Создайте объект для просмотра двух картинок, где по щелчку на кнопке происходит смена первой картинки на вторую.
6. В задании 5 добавьте кнопку для возврата к исходной картинке.
7. Создайте объект, где по щелчку на одной кнопке происходит увеличение размера картинки, а по щелчку на второй – восстановление исходного размера.
8. Создайте объект, где по щелчку на рисунке открывается новое окно размером 340×320 без строки адреса и панели инструментов с тем же рисунком большего размера (*).

9. Перепишите листинг 38 так, чтобы, не изменяя исходного массива, картинки можно было просматривать в обратном порядке.

10. Напишите скрипт, который по щелчку на кнопке случайным образом выбирал бы и отображал одно графическое изображение из 5 возможных.

11. Напишите скрипт, который по щелчку на кнопке случайным образом выбирал бы и отображал три графических изображения из трех. При выпадении одинаковых графических изображений должно выводиться сообщение методом `alert()` о совпадении (*).

Лабораторная работа 7
РАБОТА С ФОРМАМИ В JAVASCRIPT

7.1. Объект `form` и массив `forms`

Форма – это область гипертекстового документа, которая создается при помощи контейнера `<form>...</form>` и содержит элементы, позволяющие пользователю вводить информацию. Многие HTML-теги, например теги, определяющие поля ввода, области текста, контрольные переключатели, селекторные кнопки и списки, располагаются только в контейнере `<form>...</form>`. Всем перечисленным элементам в языке JavaScript соответствуют отдельные объекты (таблица 12).

Таблица 12 – **Объекты коллекции `forms`**

Объект	Описание
`button`	**Кнопка, создаваемая при помощи тега `<input type=button>`**
`reset`	**Кнопка очистки полей формы, создаваемая при помощи тега `<input type=reset>`**
`submit`	**Кнопка передачи данных, создаваемая при помощи тега `<input type=submit>`**
`radio`	**Селекторная кнопка, создаваемая при помощи тега `<input type=radio>`**
`checkbox`	**Контрольный переключатель, создаваемый при помощи тега `<input type=checkbox>`**
`text`	**Поле ввода текста, создаваемое при помощи тега `<input type=text>`**
`textarea`	**Поле ввода многострочного текста, создаваемое при помощи тега `<textarea>...</textarea>`**

password	**Поле ввода пароля, создаваемое при помощи тега <input type=password>**

Окончание таблицы 12

Объект	Описание
hidden	**Скрытое текстовое поле, создаваемое при помощи тега <input type=hidden>**
select (options[])	**Элементы <option> объекта <select>...</select>**

Программы на языке JavaScript могут обрабатывать формы, получая значения, содержащиеся в их элементах. Кроме того, данные из формы обычно передаются для обработки на удаленный Web-сервер.

Синтаксис тега </form>:

```
<form [name="formName"] [target="windowname"]
[action="serverURL"] [method="get" | "post"]
[enctype="encodingType"] [onSubmit="handlerText"]>
</form>
```

Примечание – Атрибут name – это строка, определяющая имя формы. Атрибут target задает имя окна, в котором должны обрабатываться события, связанные с изменением элементов формы. Для этого требуется наличие окна или фрейма с заданным именем.

В качестве значений атрибута target могут использоваться и зарезервированные имена _blank (документ будет загружен в новом окне), _parent (документ будет загружен в родительском окне), _self (документ будет загружен в том же окне) и _top (документ будет загружен поверх всех окон).

Атрибут action задает URL сервера, который будет получать данные из формы и запускать соответствующий скрипт. Также можно послать данные из формы по электронной почте, указав при этом значения этого атрибута, например:
mailto: ....

Формы, передаваемые на сервер, требуют задания метода передачи GET или POST, который указывается при помощи атрибута method.

Атрибут enctype задает тип кодировки MIME для посылаемых данных.

К свойствам и методам формы можно обратиться одним из способов:

```
formName.propertyName
formName.methodName (parameters)
forms[i].propertyName
forms[i].methodName (parameters)
```

Здесь formName соответствует атрибуту name объекта form, а i является целочисленной переменной, используемой для обращения к отдельному элементу массива forms, который соответствует определенному тегу <form> текущего документа.

7.2. Использование массива forms

К любой форме текущего гипертекстового документа можно обращаться как к элементу массива forms. Для этого необходимо указать индекс запрашиваемой формы.

Синтаксис:

```
document.forms[i]
document.forms['name']
```

Переменная i – номер формы в документе. Например, forms[0] – первый тег <form> в текущем документе.

Выражение вида document.forms[i] можно также присвоить переменной, например:

```
var myForm = document.forms[i];
```

Тогда, если в форме имеется поле ввода, определенное в HTML-теге <input type=text name=myField size=40>, то в программе к этому полю можно обращаться, указав имя переменной myForm и имя поля myField.

При помощи следующего оператора содержимое данного поля ввода присваивается новой переменной с именем result:

```
result = myForm.myField.value;
```

Объект form имеет шесть свойств, большинство из которых соответствуют атрибутам тега <form> (таблица 13).

Таблица 13 – **Свойства объекта form**

Свойство	Описание
action	**Соответствует атрибуту action**
elements	**Массив, содержащий все элементы формы**

encoding	**Соответствует атрибуту enctype**
length	**Количество элементов в форме**

Окончание таблицы 13

Свойство	Описание
method	**Соответствует атрибуту method**
target	**Соответствует атрибуту target**

Пример 55. Использование свойства length, значение которого соответствует количеству форм в документе:

```
var numForms = document.forms.length
```

7.3. Массив elements

Массив elements содержит все элементы HTML-формы в том порядке, в котором они определены в форме. Этот массив можно использовать для доступа к элементам формы в программе по их порядковому номеру, не используя свойства name этих элементов. Массив elements, в свою очередь, является свойством объекта forms, поэтому при обращении к нему следует указывать имя формы.

Синтаксис:
formName.elements[i]

Здесь formName может быть:

- именем объекта form, определенным при помощи атрибута name в теге <form>;
- элементом массива forms (например, forms[i], где i – номер элемента массива).

Массив elements включает данные только для чтения, т. е. динамически записать в этот объект какие-либо значения невозможно.

Объект elements имеет только одно свойство length, значением которого является количество элементов объекта form.

Выражение

```
document.forms[0].elements.length
```

возвратит количество элементов в первой форме текущего документа.

Пример 56. Создание формы с тремя элементами: поля ввода для имени и адреса и кнопки `button`. Нажатие на кнопку вызывает функцию, которая считает элементы в форме и формирует строку с информацией об объектах формы. Строка выводится на экран методом `alert()` (листинг 39, рисунок 21).

Листинг 39

```
<html>
<head>
<script language="JavaScript">
<!--
function showElem()
{
var formEl=" "
for (var n=0; n<f.elements.length; n++)
{
formEl+=n+":"+f.elements[n]+"\n"/* формируется
строка с информацией об объектах формы */
}
alert("Элементы в форме '"+f.name+"' :\n"+formEl)
}
//-->
</script>
</head>
<body>
<form name=f>
Имя:
<input type="text" size=10 name="fullname"><BR>
Адрес:
<textarea name="adr"></textarea>
<br>
<input type="button" value="Смотрим элементы"
onClick="showElem()">
</form>
</body>
</html>
```

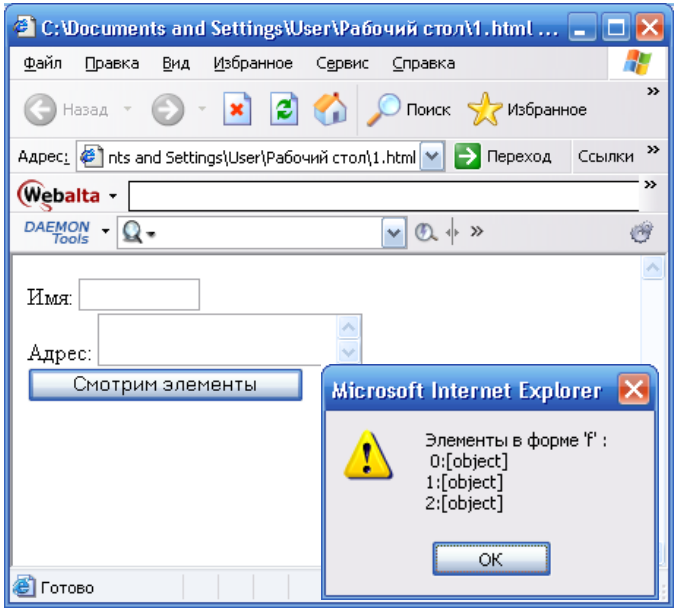


Рисунок 21 – **Результат выполнения листинга 39**

7.4. Объекты коллекции `forms`

7.4.1. Объект `checkbox`

Объект `checkbox` представляет собой контрольный переключатель. Это кнопка, которую можно установить в одно из двух состояний: включено или выключено.

Синтаксис:

```
<input [name="checkboxName"] type="checkbox" val-
ue="checkboxValue" [checked]
[onClick="handlerText"]>textToDisplay
```

Примечание – Атрибут `name` является именем объекта `checkbox`. Атрибут `value` определяет значение, которое передается серверу при пересылке значений элементов формы, если контрольный переключатель включен. Необязательный атри-

бут checked указывает, что контрольный переключатель должен быть включен по умолчанию. При помощи свойства checked можно определить, включен ли контрольный переключатель. Текст, отображаемый рядом с контрольным переключателем, задается строкой textToDisplay.

К объекту checkbox можно обращаться одним из способов:

```
checkboxName.propertyName
checkboxName.methodName(parameters)
formName.elements[i].propertyName
formName.elements[i].methodName(parameters)
```

Здесь checkboxName – значение атрибута name объекта checkbox, а formName – имя формы, которой принадлежит объект checkbox, или элемент массива forms. Также для обращения к определенному контрольному переключателю следует использовать его порядковый номер (i) в массиве всех элементов формы.

Для объекта checkbox предусмотрен только один обработчик событий – onClick.

7.4.2. Объект password

Объект password представляет собой поле ввода, содержимое которого не подлежит просмотру. Вместо каждого символа, введенного в таком поле, отображается символ звездочки (*). В то же время содержимое данного поля сохраняется как обычный текст.

Синтаксис:

```
<input type="password" [name="passwordName"]
[size=integer][value="textValue"]>
```

Синтаксис этого тега такой же, как и у тега, задающего поле ввода, однако атрибут type имеет значение "password". Значением данного атрибута для поля ввода всегда является строка "text".

Для обращения к свойствам и методам поля пароля применяются выражения следующего вида:

```
passName.propertyName
passName.methodName(parameters)
formName.elements[i].propertyName
formName.elements[i].methodName(parameters)
```

Здесь passName – значение атрибута name объекта password, а formName – имя формы, которой принадлежит объект password, или элемент массива forms.

Свойства объекта password представлены в таблице 14.

Таблица 14 – **Свойства объекта password**

Свойство	Описание
defaultValue	**Значение по умолчанию, задаваемое с помощью атрибута value**
name	**Соответствует атрибуту name**
value	**Соответствует текущему значению объекта password**
type	**Значением этого свойства для всех объектов password является строка "password"**

Для объекта password применяются обработчики событий: onFocus, onBlur, onSelect.

7.4.3. Объект radio

Объект radio представляет собой селекторную кнопку, определяемую в HTML-форме. Селекторные кнопки позволяют выбрать только один из предлагаемых вариантов.

Синтаксис:

```
<input type="radio" name="radioName" val-
ue="buttonValue" [checked]
[onClick="handlerText"]> textToDisplay
```

Примечание – В атрибуте name задается имя селекторной кнопки. Для селекторных кнопок, принадлежащих к одной группе, необходимо указывать одинаковые значения атрибута name. Значение атрибута value возвращается серверу при передаче формы. С помощью атрибута checked можно выбрать селекторную кнопку, которая будет активизирована по умолчанию. В качестве значения атрибута onClick указывают обработчик события, связанного с выбором селекторной кнопки. Текст, описывающий вариант выбора селекторной кнопки, задается в строке textToDisplay.

Для обращения к методам и свойствам селекторной кнопки используют следующие выражения:

```
radioName[i].propertyName
radioName[i].methodName(parameters)
formName.elements[i].propertyName
formName.elements[i].methodName(parameters)
```

Здесь `radioName` – значение атрибута `name` объекта `radio`, а `formName` – имя формы, которой принадлежит объект `radio`, или элемент массива `forms`.

Для обращения к селекторным кнопкам можно использовать массив элементов формы `elements`. Например, к первой селекторной кнопке с именем `radio1`, принадлежащей первой форме текущего документа, можно обратиться при помощи выражения

```
document.forms[0].radioName[0]
```

Свойства объекта `radio` представлены в таблице 15.

Таблица 15 – **Свойства объекта `radio`**

Свойство	Описание
`checked`	**Содержит булевое значение `true` или `false` в зависимости от того, выбрана или нет данная селекторная кнопка**
`defaultChecked`	**Соответствует атрибуту `checked`, содержит булевое значение**
`length`	**Представляет количество селекторных кнопок в объекте `radio`**
`name`	**Соответствует атрибуту `name`**
`value`	**Соответствует атрибуту `value`**
`type`	**Для объектов `radio` значением этого атрибута является строка `"radio"`**

Объект `radio` имеет обработчик событий `onClick`, который используется для обработки событий, связанных с активизацией селекторной кнопки.

7.4.4. Объект `reset`

Объект `reset` отображается в HTML-форме как кнопка перезагрузки, которая возвращает каждый элемент формы к его исходному состоянию, а значения по умолчанию устанавливаются при помощи

атрибута value.

Синтаксис:
```
<input type="reset" [name="resetName"] val-
ue="buttonText" [onClick="handlerText"]>
```

Примечание – Атрибут name задает имя объекта reset, а атрибут value – текст, отображаемый на кнопке.

Для обращения к методам и свойствам объекта reset используется следующий синтаксис:

```
resetName.propertyName
resetName.methodName(parameters)
formName.elements[i].propertyName
formName.elements[i].methodName(parameters)
```

Здесь resetName – значение атрибута name объекта reset, а formName – имя формы, которой принадлежит объект reset, или элемент массива forms.

Свойства объекта reset представлены в таблице 16.

Таблица 16 – **Свойства объекта reset**

Свойство	Описание
name	**Соответствует атрибуту name**
value	**Соответствует атрибуту value**
type	**Для объектов reset всегда имеет значение "reset"**

Для объекта reset можно определить обработчик события onClick.

7.4.5. Объект select и массив options

Объект select отображается как поле со списком и является элементом формы, определенной в HTML-документе. Элементам списка, заданного в объекте select, соответствует массив options. Объект select представляет собой свойство объекта form, в то время как массив options является свойством объекта select.

Синтаксис тега <select>:
```
<select name="selectName" [OnBlur="handlerText"]
```

```
[OnChange="handlerText"] [OnFocus="handlerText"]>
<option value="optionValue" [selected]>
textToDisplay
</select>
```

Примечание – Атрибут `name` задает имя объекта `select`. Контейнер `<select>...</select>` не используется без тегов `<option>`, поскольку эти теги задают элементы списка. Атрибут `value` тега `<option>` представляет значение, которое пересылается серверу при передаче формы. Атрибуту `value` соответствует свойство объекта `select`. Список может содержать элементы, выбранные по умолчанию, что задается при помощи атрибута `selected`. Текст для элементов списка необходимо задать в строке `textToDisplay`. Каждому элементу списка соответствует значение, устанавливаемое в атрибуте `value` тега `<option>`.

Для обращения к свойствам и методам объекта `select` используются выражения следующего типа:

```
selectName.propertyName
selectName.methodName(parameters)
formName.elements[i].propertyName
formName.elements[i].methodName(parameters)
```

Здесь `selectName` – значение атрибута `name` объекта `select`, а `formName` – имя формы, которой принадлежит объект `select`, или элемент массива `forms`.

К отдельным элементам списка, определенного в объекте `select`, можно обратиться при помощи массива `options` или массива `elements`, используя выражения следующего вида:

```
selectName.options[i].propertyName
formName.elements[i].options[index].propertyName
```

Здесь `selectName` – имя, заданное в атрибуте `name` тега `<select>`, а `formName` – имя формы, в которой определен данный объект `select`.

Свойства объекта `select` представлены в таблице 17.

Таблица 17 – **Свойства объекта `select`**

Свойство	Описание
`length`	**Количество тегов `<option>`, заданных в теге `<select>`**

name	**Соответствует атрибуту name**
options	**Массив значений тегов <option>**
selectIndex	**Содержит индекс выбранного элемента, а если выбрано несколько элементов, то индекс первого**
type	**Для списков с возможностью выбора одного элемента содержит значение "select-one", а для списков с возможностью выбора нескольких элементов – "select-multiple"**

Свойства массива options представлены в таблице 18.

Таблица 18 – **Свойства массива options**

Свойство	Описание
defaultSelected	**Соответствует первому тегу <option>, определенному с атрибутом selected**
index	**Номер элемента в массиве**
length	**Количество элементов в списке объекта select**
selected	**Не равное нулю, если выбран данный элемент списка**
selectedIndex	**Содержит индекс выбранного элемента**
value	**Соответствует атрибуту value**

Для объекта select можно определить обработчики событий onBlur и onChange, задавая реакцию объекта на события, связанные с потерей и получением фокуса ввода.

Пример 57. Выбор времени года (листинг 40, рисунок 22).

Листинг 40

```
<html>
<head>
<title>Времена года</title>
</head>
<body>
<script language = "JavaScript">
<!--
function showSelected()
{
i = t_year.season.selectedIndex /* индекс
    выбранного элемента в форме t_year*/
n = t_year.season.options[i].text /* текст,
    который указан в теге <option>*/
alert("Выбрана опция: "+ i + "\n"
```

```
+"Текст выбранной опции: "+ n)
}
//-->
</script>
<form name ="t_year">
Какое время года лучше?
<select name = "season">
<option>Зима
<option>Весна
<option>Лето
<option>Осень
</select>
<p>
<input type = "button" value="Смотрим что выбрали"
onClick="showSelected()">
</form>
</body>
</html>
```

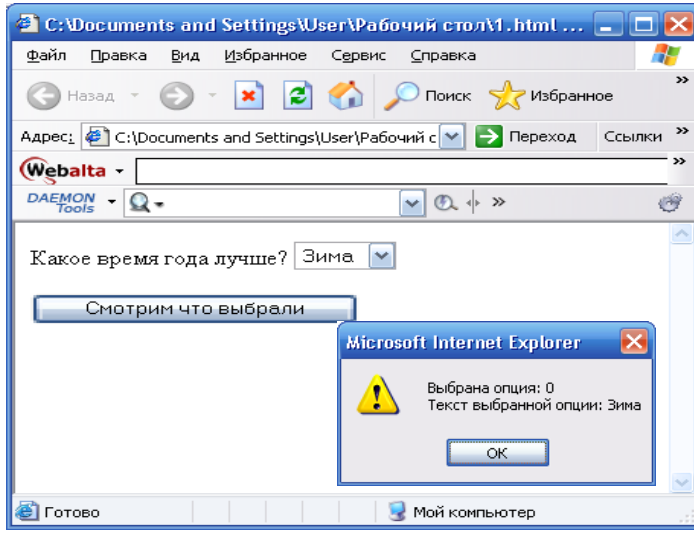


Рисунок 22 – **Выбор времени года**

7.4.6. Объект `submit`

Объект submit отображается как кнопка передачи данных из формы на сервер. Имя файла-обработчика, принимающего данные, задано при помощи атрибута action тега <form>.

Синтаксис:

```
<input type="submit" name="submitName" val-
ue="buttonText" [onClick="handlerText"]>
```

Примечание – С помощью атрибута name задается имя объекта submit, которое является значением одноименного свойства объекта. В качестве значения атрибута value используется строка текста, отображаемая на кнопке.

Для обращения к методам и свойствам объекта submit применяются следующие выражения:

```
submitName.propertyName
submitName.methodName(parameters)
formName.elements[i].propertyName
formName.elements[i].methodName(parameters)
```

Здесь submitName – значение атрибута name объекта submit, а formName – имя формы, которой принадлежит объект submit, или элемент массива forms.

Для объекта submit можно определить обработчик события onClick.

7.4.7. Объект `text`

Объект text – это поле ввода, определяемое в теге <input> и предоставляющее пользователю возможность вводить текстовые данные. Объекты text содержат данные, которые можно читать и динамически изменять.

Синтаксис тега:

```
<input type="text" name="textName" [val-
ue="textValue"] [size=integer]
[onBlur="handlerText"] [onChange="handlerText"]
[onFocus="handlerText"] [onSelect="handlerText"]>
```

Примечание – С помощью атрибута name задается имя объекта text. Значение атрибута value определяет содержимое поля по умолчанию. Атрибут size

устанавливает размер поля в символах.

Для обращения к методам и свойствам объекта `text` используют выражения следующего вида, типичные для всех элементов формы:

```
textName.propertyName
textName.methodName(parameters)
formName.elements[i].propertyName
formName.elements[i].methodName(parameters)
```

Здесь `textName` – значение атрибута `name` объекта `text`, а `formName` – имя формы, которой принадлежит объект `text`, или элемент массива `forms`.

Свойства объекта `text` представлены в таблице 19.

Таблица 19 – **Свойства объекта `text`**

Свойство	Описание
`defaultValue`	**Соответствует атрибуту `value`**
`value`	**Текущее значение объекта `text`**
`name`	**Соответствует атрибуту `name`**
`type`	**Соответствует атрибуту `type` и содержит значение `"text"`**

Для объекта `text` можно определить четыре обработчика событий: `onBlur`, `onChange`, `onFocus` и `onSelect`.

Пример 58. Проверка корректности e-mail. Будем считать e-mail корректным, если строка не пустая и содержит символ «@» (листинг 41, рисунки 23–25).

Листинг 41

```
<html>
<head>
<title>Проверка</title>
<script language="JavaScript">
<!--
function test()
{
if (first.t.value== "" ||
    first.t.value.indexOf('@', 0) == -1)
alert("Неверно введен адрес e-mail!")
else
```

```
alert("OK!")
}
// -->
</script>
</head>
<body>
<form name="first">
Введите Ваш адрес e-mail:
<br>
<input type="text" name="t">
<input type="button" name="button2"
value="Проверка" onClick="test()">
</body>
</html>
```

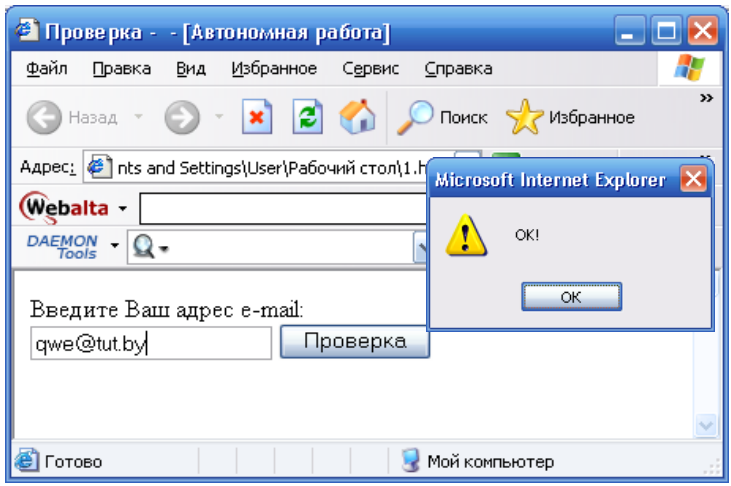


Рисунок 23 – **Положительный результат проверки данных**

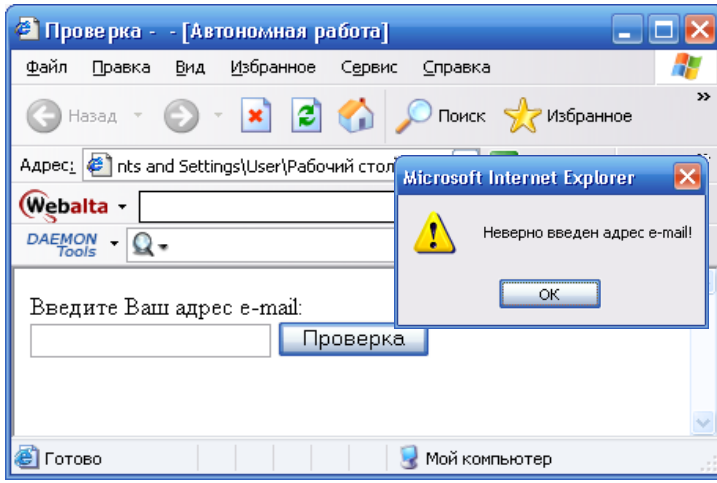


Рисунок 24 – **Отрицательный результат проверки (поле пустое)**

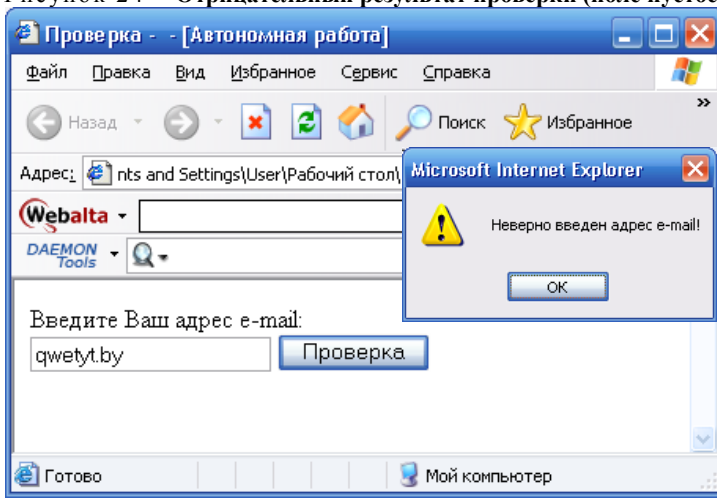


Рисунок 25 – **Отрицательный результат проверки (в электронном адресе отсутствует знак «@»)**

7.4.8. Объект `textarea`

Объект `textarea` соответствует области текста, определенной в форме. Элементы этого типа используются для ввода нескольких строк текста в свободном формате.

Синтаксис:

```
<textarea name="textareaName" rows="integer"
cols="integer" [onBlur="handlerText"]
[onChange="handlerText"] [onFocus="handlerText"]
[onSelect="handlerText"]>[textToDisplay]
[wrap="hand | soft"]
</textarea>
```

Примечание – Атрибут `name` определяет имя области текста. Атрибуты `rows` и `cols` задают размеры поля области в символах. Строка `textToDisplay` представляет собой необязательный текст, помещенный в область текста при первом отображении на экране. Значение атрибута `wrap` определяет, каким образом введенный в поле текст разбивается на строки – значение `soft` задает отображение строк в области текста полностью, в противном случае текст между двумя символами конца строки размещается в одной строке.

Для обращения к методам и свойствам объекта `textarea` применяются типичные для элементов формы выражения:

```
textareaName.propertyName
textareaName.methodName(parameters)
formName.elements[i].propertyName
formName.elements[i].methodName(parameters)
```

Здесь `textareaName` – это значение атрибута `name` объекта `textarea`, а `formName` – имя формы, которой принадлежит объект `textarea`, или элемент массива `forms`.

Содержимое объектов `textarea` может динамически изменяться путем присваивания нового значения их свойству `value`. Например:

```
document.forms[0].myArea.value = "Новый текст"
```

Свойства объекта `textarea` представлены в таблице 20.

Таблица 20 – **Свойства объекта `textarea`**

Свойство	Описание
`defaultValue`	**Значение содержит текст, помещенный в контейнер `<textarea>...</textarea>`**
`name`	**Соответствует атрибуту `name`**

Свойство	Описание
value	**Соответствует текущему значению объекта textarea**
type	**Для объекта textarea содержит значение "textarea"**

Для объекта textarea можно определить четыре обработчика событий: onBlur, onChange, onFocus и onSelect.

Пример 59. Использование обработчика событий onChange для обработки данных, введенных в текстовое поле (листинг 42, рисунки 26 и 27).

Листинг 42

```
<html>
<head>
</head>
<body>
<script language = "JavaScript">
<!--
function sCange()
{
alert ("Содержимое текстовой области изменено")
}
//-->
</script>
<form>
Измените этот текст и перейдите в другое поле формы:
<br>
<textarea name="tarea" rews=5 cols=40
onChange="sCange()">
Это объект textarea
</textarea>
<br>
<input type="text" size=35 name="stxt">
</form>
</body>
</html>
```

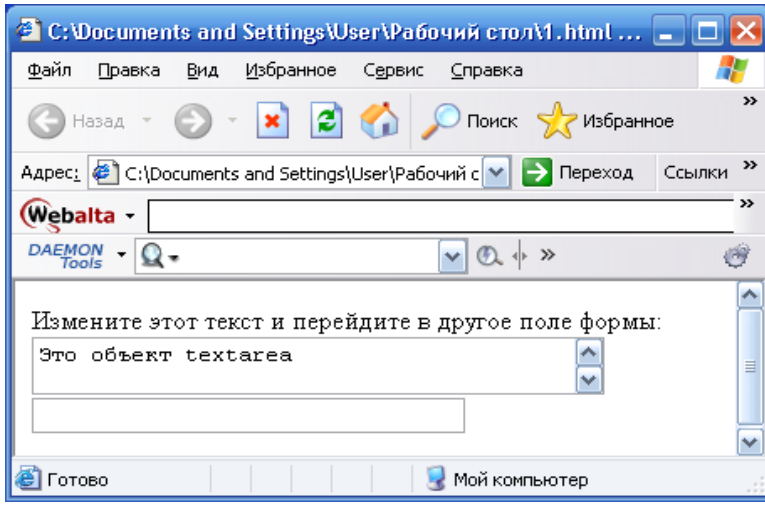


Рисунок 26 – **Исходные данные для использования обработчика событий `onChange`**

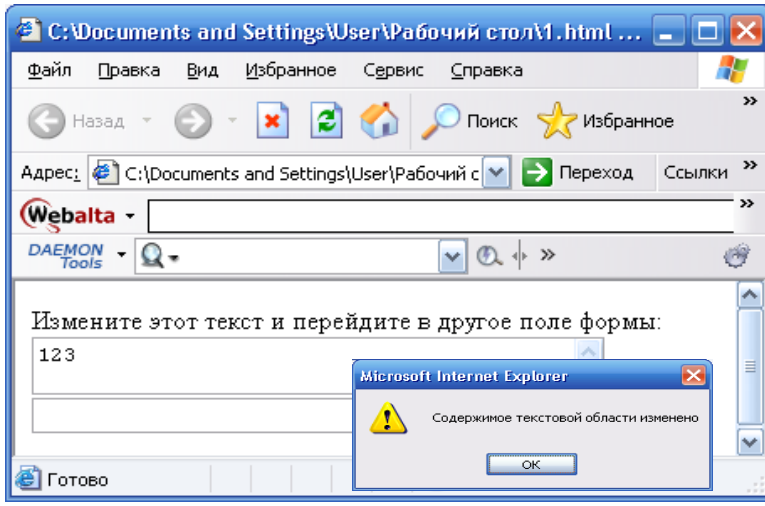


Рисунок 27 – **Результат использования обработчика событий `onChange` выведен методом `alert()`**

7.5. Изменение значений атрибутов полей формы

До сих пор рассматривались примеры, в которых использовались

значения различных полей ввода. В программах на JavaScript можно текстовым полям присваивать новые значения путем изменения атрибута `value`.

Пример 60. Создание формы с тремя полями для ввода чисел *a*, *b*, *c* и одной кнопкой. Вычисление значения *s* по формуле $s = a + b - c$. Вывод результата в четвертом поле (листинг 43, рисунок 28).

Листинг 43

```
<html>
<head>
<title>Вычисления</title>
<script language="JavaScript">
<!--
function test()
{
a=parseFloat(first.text1.value)
b=parseFloat(first.text2.value)
c=parseFloat(first.text3.value)
s=a+b-c
first.text4.value=s
}
// -->
</script>
</head>
<body>
<form name="first">
Введите первое число:
<input type=text name="text1"><br>
Введите второе число:
<input type=text name="text2"><br>
Введите третье число:
<input type=text name="text3"><br>
<input type="button" value="Вычислить"
onClick="test()">
<br>
Результат:
<input type=text name="text4"><br>
</body>
</html>
```

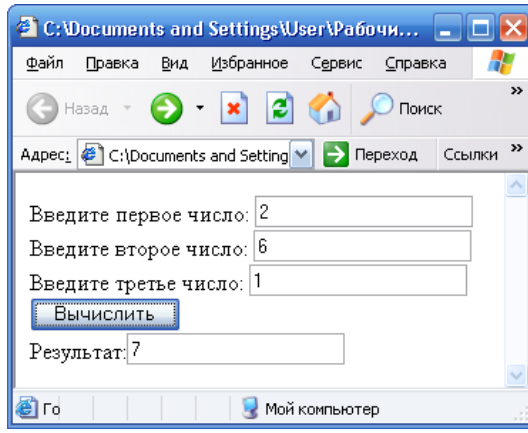


Рисунок 28 – **Результат выполнения листинга 43**

Пример 61. Анкета для покупателя по оценке работы интернет-магазина (листинг 44, рисунок 29).

Листинг 44

```
<html>
<head>
<title>анкета покупателя</title>
<script language="JavaScript">
<!--
var k=1
var s=""
function mov()
{
n=form1.data.value
s=s+k+' '+form1.data[n].text+"\r\n"
k=k+1
form1.res.value=s
}
function ref()
{
k=1
s=""
}
// -->
</script>
</head>
```

```
<body>
<h2>
<font color="#800000">Интернет магазин ABC
www.abc.by</font>
</h2>
<h3>
<font color="#0000ff">Перечислите, в порядке
важности, что Вас привлекает в работе нашего
магазина (наведите курсор на строчку и нажмите
левую клавишу)</font>
</h3>
<form name="form1">
<table border="6" cellpadding="7" cellspacihg="0"
width="100">
<tr>
<td>
<select name="data" size="8" OnChange="mov()">
<option value=0> низкие цены
<option value=1> высокое качество товаров
<option value=2> удобный способ оплаты
<option value=3> быстрота доставки купленного
                 товара
<option value=4> легкость нахождения сайта
<option value=5> удобство и простота заказа
                 товаров
</select>
</td>
<td align="bottom" width="95%">
<textarea name="res" rows="8" cols="30">
</textarea>
</td></tr>
</table><br>
<input type="reset" value="обновить"
onClick="ref()">
</form>
</body>
</html>
```

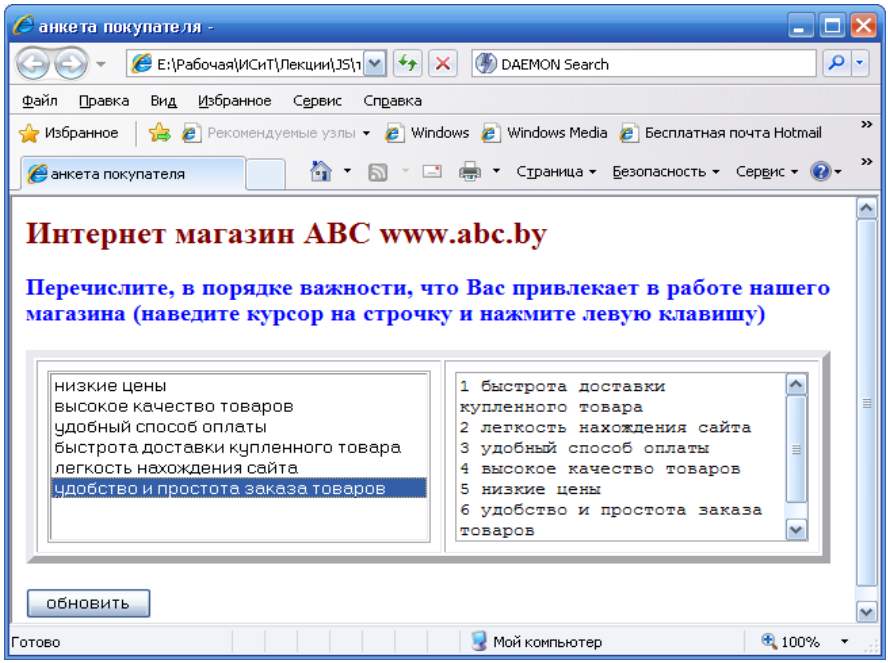


Рисунок 29 – **Результат выполнения листинга 44**

Задания для самостоятельного выполнения

1. Выполните скрипты, приведенные в листингах 39–44.

2. Создайте форму для регистрации, используя следующие поля: ФИО, пол, адрес, e-mail, телефон, вопросы с вариантами ответа, комментарии, кнопку проверки данных. Проверьте соответствующие поля на наличие и корректность введенных данных.

3. Допишите задание 1 так, чтобы в случае некорректно введенных данных курсор оставался в поле ввода до тех пор, пока данные не будут введены корректно. Рекомендуется использовать метод `focus()` для возврата фокуса в текущее текстовое поле (*).

4. Пусть требуется рассчитать заработок членов бригады из пяти человек. Бригада за выполнение работы получила 1 000 усл. ед. Курс валюты составляет 2 900 р. за 1 усл. ед. Доплата бригадиру – 50 000 р.

Заработок члена бригады (A) вычисляется по формуле

$$A = \frac{S \cdot K - D}{N},$$

где S – суммарный заработок бригады в условных единицах;

K – курс валюты (рублей за 1 усл. ед.);
D – доплата бригадиру в рублях;
N – количество членов бригады.

Заработок бригадира (B) вычисляется по формуле

$$B = A + D.$$

Создайте форму со следующими элементами:

- с полями *для ввода* суммарного заработка бригады в условных единицах, курса валюты (рублей за 1 усл. ед.), доплаты бригадиру в рублях, количества членов бригады;
- с полями *для вывода* заработка членов бригады, заработка бригадира;
- с кнопками для вычисления зарплаты в рублях членов бригады, очистки полей формы.

5. Напишите скрипт, позволяющий выбирать цвет фона из списка. Используйте обработчик событий `OnChange`.

6. Реализуйте простейший калькулятор, выполняющий основные арифметические операции для двух чисел (*).

СПИСОК РЕКОМЕНДУЕМОЙ ЛИТЕРАТУРЫ

1. **Вилтон, П.** JavaScript. Руководство программиста / П. Вилтон, Дж. МакПик. – СПб. : Питер, 2009. – 720 с.

2. **Дунаев, В. В.** Самоучитель JavaScript / В. В. Дунаев. – 3-е изд., перераб. и доп. – СПб. : Питер, 2008. – 400 с.

3. **Дунаев, В. В.** Сценарии для Web-сайта: PHP и JavaScript / В. В. Дунаев. – 2-е изд., перераб. и доп. – СПб. : БХВ-Петербург, 2008. – 576 с.

4. **Колисниченко, Д. Н.** Современный сайт на PHP и JavaScript / Д. Н. Колисниченко. – СПб. : Питер, 2009. – 176 с.

5. **Прохоренок, Н. А.** HTML, JavaScript, PHP и MySQL. Джентльменский набор Web-мастера / Н. А. Прохоренок. – 2-е изд., перераб. и доп. – СПб. : БХВ-Петербург, 2009. – 880 с.

СОДЕРЖАНИЕ

Учебное издание

ИНФОРМАЦИОННЫЕ СИСТЕМЫ И ТЕХНОЛОГИИ

WEB-ПРОГРАММИРОВАНИЕ НА JAVASCRIPT

Практикум
к лабораторным занятиям
для студентов специальности 1-26 03 01
«Управление информационными ресурсами»

Автор-составитель
Бондарева Валентина Викторовна

Редактор М. П. Герасенко
Технический редактор И. А. Козлова
Компьютерная верстка Н. Н. Короедова

Подписано в печать 14.12.11. Бумага типографская № 1.
Формат 60 × 84 $^{1}/_{16}$. Гарнитура Таймс. Ризография.
Усл. печ. л. 6,04. Уч.-изд. л. 5,90. Тираж 100 экз.
Заказ №

Учреждение образования
«Белорусский торгово-экономический университет
потребительской кооперации».
246029, г. Гомель, просп. Октября, 50.
ЛИ № 02330/0494302 от 04.03.2009 г.

Отпечатано в учреждении образования
«Белорусский торгово-экономический университет
потребительской кооперации».
246029, г. Гомель, просп. Октября, 50.

БЕЛ[illegible]СОЮЗ
УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ
«БЕЛОРУССКИЙ ТОРГОВО-ЭКОНОМИЧЕСКИЙ
УНИВЕРСИТЕТ ПОТРЕБИТЕЛЬСКОЙ КООПЕРАЦИИ»

Кафедра информационно-вычислительных систем

ИНФОРМАЦИОННЫЕ СИСТЕМЫ И ТЕХНОЛОГИИ

WEB-ПРОГРАММИРОВАНИЕ НА JAVASCRIPT

Практикум
к лабораторным занятиям
для студентов специальности 1-26 03 01
«Управление информационными ресурсами»

Гомель 2011